A Simplification of a Real-Time Verification Problem

Indranil Saha[1] Janardan Misra[2]
Suman Roy[3]
[1] Comp. Sc. Dept., Uni. of California, Los Angeles, CA 90095, USA.
Email:indranil@cs.ucla.edu
[2] HTS Research, BG Road, Bangalore 560076, India.
Email: janmishra@gmail.com
[3] SETLABS, Infosys Tech. Ltd., #44 Electronic City, Bangalore 560100, India.
Email: suman_roy@infosys.com

August 10, 2010

Abstract

We revisit the problem of real-time verification with dense time dynamics using timeout and calendar based models, originally proposed by Dutertre and Sorea, and simplify this to a finite state verification problem. To overcome the complexity of verification of real-time systems with dense time dynamics, Dutertre and Sorea, proposed timeout and calender based transition systems to model the behavior of real-time systems and verified `safety` properties using k-induction in association with bounded model checking. In this work, we introduce a specification formalism for these models in terms of Timed Transition Diagrams and capture their behavior in terms of semantics of Timed Transition Systems. Further, we discuss a technique, which reduces the problem of verification of qualitative temporal properties on infinite state space of (a large fragment of) these timeout and calender based transition systems into that on clockless finite state models through a two-step process comprising of digitization and canonical finitary reduction. This technique enables us to verify `safety` invariants for real-time systems using finite state model-checking avoiding the complexity of infinite state (bounded) model checking and scale up models without applying techniques from induction based proof methodology. Moreover, we can verify `liveness` properties for real-time systems, which is not possible by using induction with infinite state model checkers. We present examples of Fischer's Protocol, Train-Gate Controller, and TTA start-up algorithm to illustrate how such an approach can be efficiently used for verifying `safety`, `liveness`, and `timeliness` properties specified in LTL using finite state model checkers like SAL-smc and Spin. We also demonstrate how advanced modeling concepts like inter-process scheduling, priorities, interrupts, urgent and committed location can be specified as extensions of the proposed specification formalism, that can be subjected to the proposed two step reduction technique for verification purposes.

Keywords: Real-Time Systems; Timeout and Calendar Model; Clockless Model; Finite State Verification

Contents

1 Introduction

Real-time systems are an important class of mission critical systems, which have been well studied for their design, implementation, performance and verification. Modeling and verification of real-time systems in dense time domain is an important problem area that evoked lot of research interest in the recent past. Because of the fact that the state space of real-time systems with continuous dynamics is uncountable, modeling and verification of them is rather difficult, in particular using explicit state model checkers. Many formalisms have been used to model and verify real-time systems. Notable among them are different kinds of timed transition models [Alu99, HMP92b], timed process algebras [BeJ91, DaS95, NiS94], and real-time logics [AlH91, BMN00].

In [DuS04a], Dutertre and Sorea, considered verification of a train-gate controller modeled as a timed automata. Though they could specify the timed automata model in terms of state transition system in infinite state model checker SAL [MOR04], it however did not to produce the desired results. In particular, the clock variables occurring in timed automata would be updated in arbitrarily small increments leading to infinite trajectories during which the discrete state remained idle. This made proof of `safety` properties by k-induction quite hard, and sometimes impossible. The fact that the traditional semantics of timed automata allows several time steps to occur in succession is an obstacle in proving properties by k-induction.

To address this problem the same authors proposed timeout and calender based transition models, [DuS04a, DuS04b], originally from discrete event simulation, to represent the behavior of timed

triggered systems with dense time dynamics. These models are amenable to general-purpose verification environments, like SAL in which state machines and their compositions can be specified. In this modeling approach, each process in the system has a timeout that holds the time when the next discrete transition of the process would happen, and there is a global data structure, called *calendar*, which stores future events (message delivery) and the time points at which these events are scheduled to occur. During the time progress transition, time is advanced to the minimum of timeouts of processes, or to the least time point at which a message will be delivered in future, whichever is less. Further, Dutertre and Sorea, used this calendar based model along with timeouts for individual processes to model TTA startup protocol in SAL [DuS04b]. Using bounded model checking, they proved a `safety` property by k induction. However, these proofs using k-induction do not usually scale up well; a `safety` property often cannot be proved at induction depth 1. Sometimes `safety` properties are not at all inductive and need the support of auxiliary lemmas. In [DuS04b], a `safety` property for the TTA startup algorithm with only 2 nodes has been proved by using 3 additional lemmas. A verification diagram based abstraction method proposed in [Rus00] has been used to prove the same `safety` (invariant) property for scaled up models (upto 10 nodes). However, `liveness` properties still remain beyond the scope of this approach.

While only `safety` properties can be verified on these models with dense time, discrete time modeling of the same can help verify `liveness` and `timeliness` properties, and also help scale up proofs for `safety` properties. It turns out that verification of a real-time system in dense domain is equivalent to verifying the system in discrete domain if both the behavior of the system captured by the model and the properties considered are digitizable [HMP92a]. It can be shown that if the timeout updates are not restricted to $(0, 1)$-intervals, then similar to the timed transition system of [HMP92a] (refer to theorem 2 therein), transition systems for timeout and calendar based models also give rise to digitizable behaviors (computations). Also verification of qualitative properties like `safety` and `liveness`, in discrete time domain is equivalent to verifying these properties in dense time domain (refer to proposition 1 in [HMP92a]).

Techniques like bounded model checking [MRS03, DuS04a] can be useful for detecting bugs during the verification process even in discrete domain, where one systematically searches for counterexamples of length bounded by some integer k. The bound k is increased until a bug is found, or some pre-computed completeness threshold is reached. Unfortunately, it is usually very expensive to compute completeness thresholds. Also these thresholds may be too large to effectively explore the bounded search space. Additionally, such completeness thresholds may be absent for many infinite-state systems. A finite state modeling of the system can help exploring the state space much easily. Examples of finite state model checkers are Spin [Hol93], SAL-smc solvers [DuS04a] etc. Spin has been used to finitely model TTA startup algorithm using a clockless calendar based model [SMR07]. In terms of scalability, finite state verification of TTA in Spin is almost comparable to the verification of TTA based on verification diagram oriented abstraction method [DuS04b]. Moreover, `liveness` properties can be verified in this framework.

In this work, we aim to carry out a finite state modeling and verification on timeout and calendar models without continuously varying clocks. As there are drawbacks of those models earlier proposed from the point of view of design considerations, like absence of formally defined syntactic models and associated semantics, we slightly deviate from them. We consider the specification framework of timed transition diagrams and extend it to formalize timeout and calendar based models as timeout and calendar based transition diagrams and their behavior in terms of semantics of transition systems. The benefits that we derive from using this formalization are many-fold. Our framework of timeout transition diagrams inherits most of the properties of classical timed transition system introduced in [HMP92b]. Most of the techniques, like digitization that can be applied to these timed transition systems are applicable to our formalization also. This can be also used to model time-triggered systems and reason about them. Finally we use this formal modeling framework to reduce continuous time verification problem to discrete time finite state verification, albeit under some restrictions. Towards that, we use a two step technique comprising of digitization and finitary reduction (a schematic diagram of this technique is shown in Figure 1). We show that the computations of timeout and calendar models are digitizable provided the timeout increments are not restricted to $(0, 1)$-interval. As LTL properties are qualitative and hence, are digitizable, verification of LTL properties on timeout and calendar models in dense time is equivalent to that in discrete time. The next step is to reduce this problem into an equivalent finite state verification problem. We could not directly proceed to extract finite state models from dense time models, since the latter models are inherently infinite (and dense) and hence it is not possible to render them finite even by bounding the variables. Also note that such a modeling cannot be directly subjected to finite state verification since for timeout and calendar based models, global time and timeouts always increase. Nonetheless, we propose a finitary reduction technique which effectively reduces the infinite state timeout

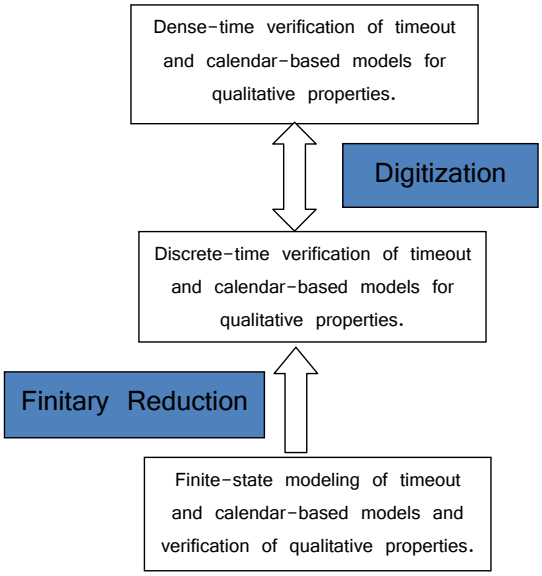

Figure 1: A two-step verification process

and calendar based transition systems with discrete dynamics into a finite state transition system. We achieve this by using a clockless modeling technique which effectively strips the model of the global clock and keeps track of the relative updation of timeouts, and restricts the values of variables/timeout updates to bounded domains. We demonstrate by examples, how such a modeling approach can be efficiently used for verifying safety, liveness, and timeliness properties using finite state model checkers, SAL-smc and Spin. We also highlight the scalability of such models for verification purposes by comparing the performance of such models under dense time and finite state modeling. A preliminary version of this paper appeared in [SMR07].

The remainder of the paper is organized as follows. In the next section, we briefly discuss the timeout and calendar based modeling as presented in [DuS04a, DuS04b]. In Section 3, we present the formalization of these models in terms of timeout transition diagrams and their behavior in terms of the semantics of transition systems. We discuss the technique of digitization in Section 5 and present our first step of reduction of dense-time verification problem to integral time verification problem. In Section 6, we describe the finitary reduction technique and subsequently, formalize it in terms of clockless modeling. We present experimental results in Section 7. A few extensions of our framework are described in Section 8 followed by concluding remarks in Section 9.

1.1 Related Work

There have been earlier attempts to model and verify time-triggered systems using extensions of finite state model checkers, e.g., Spin. Spin [Hol93] is a tool for automatically model checking distributed systems, but it does not allow explicit representation for time. There are mainly two attempts for extending Spin with time. Real-time extension of Spin (RT-Spin [TrC96]) is one such attempt, that makes use of timed Buchi automata [AlD94] with real-valued clocks as a modeling framework. However, this formalism is incompatible with the partial order reduction which is supported by Spin. Another is the work on DT-Spin [BoD98a, BoD98b], which allows one to quantify (discrete) time elapsed between events, by specifying the time slice in which they occur. DT-Spin is compatible with the partial order reduction and has been used to verify industrial protocols, like, AFDX Frame management protocol [SaR06a] and TTCAN [SaR06b]. Nonetheless, systems with asynchronous communication with bounded delays between components cannot be modeled directly by using the kind of asynchronous channels that Spin provides, since there is no explicit provision to capture message transmission delays. A possible way is to model each channel as a separate process with delay as a state variable. In [BoD98a], the channels in the example of PAR protocol have been implemented in the same way. But for systems with relatively large number of components and high degree of connectivity among them, modeling channels in this way is difficult, and hence state space explosion becomes an unavoidable problem.

The concept of clockless modeling has been introduced in [Pik05]. In that Pike builds on the work

of [DuS04a] and proposes a new formalism called Synchronizing Timeout Automata (STA) to reduce the induction depth k required for k-induction. He introduces a clockless semantics for STA so that the resulting transition system does not involve a clock. STA in effect, describes the overall system architecture in terms of timeout transition system introduced in [DuS04a]. A closer analysis of the SAL model for the example of Train-Gate Controller presented in [Pik05], reveals that the considered model is not deadlock free. This is because the model fails to specify the timeout updation rules precisely for the transitions leading to a waiting state. When a process is waiting for an external signal, its timeout should be set to a value greater than the current value of the timeouts of the senders of the expected signal. This kind of modeling errors could possibly be eliminated with a suitable modeling framework such as the one proposed in this paper.

To our knowledge, the first attempt to convert TA to untimed TA is taken up in [AlD94]. Building upon these, in [ChH04] the authors discuss how a special kind of model for specifications written in Duration Calculus (DC) [CHR91] can be generated in which, DC formulas would correspond to regular expressions over a state of special symbols. The models for DC formulas contain discrete states and digitization of continuous states, thereby enabling reasoning in a single framework of both discrete and continuous time. Applying discretization on the continuous component of real-time systems, these models could be further translated into Promela models for verification experiments using SPIN.

2 Timeout and Calendar-based Real-Time Models

In this section we briefly discuss the timed automata [AlD94], timeout, and calendar-based models introduced earlier in [DuS04b].

2.1 Timed Automata

Timed automata (TA) was introduced by Alur et al. in [AlD94] as a clock based model for specifying real-time system designs. TA is widely used for modeling and verification of real-time systems. Many tools are available for analyzing timed automata e.g., UPPAL [BDL04], Kronos [Bozga], Rabbit [BLN03]. For further details on TA, the reader is referred to [AlD94].

2.2 Timeout Transition Model

Dutertre and Sorea [DuS04a, DuS04b] used timeout based modeling to formally verify real-time systems using $k-$induction in SAL model checker. A Timeout Transition Model (TTM), which is a model of the combined system behavior, contains a finite set of timeouts and a global clock variable t. Timeouts define the time points when discrete transitions will be enabled in the future. The clock variable t keeps track of the current time. In practice a typical real-time system may contain a number of processes. Every process is associated with one timeout which records the future point of time when the next discrete transition for the process is scheduled to occur. Transitions in this model are classified into two types: time progress transitions and discrete transitions. In time progress transition, t (time) advances to the minimum valued timeout(s). Discrete transition occurs when t is equal to the minimum valued timeout(s). If there are more than one processes, which have their timeouts equal to the minimum value, one of them is randomly chosen and corresponding discrete transition occurs updating the value of the timeout for the selected process. Timeout based modeling approach is suited to model systems where the processes communicate via shared variables or the communication between the processes is a rendezvous one.

2.3 Calendar Transition Model

Interprocess communication delay during message transfers cannot be modeled using timeout based modeling because delays are beyond the control of individual processes. Addition of an *event calendar*, a globally shared data structure, is proposed as a convenient way to model such delays [DuS04b]. This model is called Calendar Transition Model (CTM). A *calendar* is a set of bounded size of the form $C = \{\langle e_1, t_1\rangle, \ldots, \langle e_r, t_r\rangle\}$, where each event e_i is associated with the time point t_i when it is scheduled to occur. There is fundamental difference between a clock and a calendar in the sense that while the former measures the time elapse since its last reset, the latter stores expected delivery delays for all undelivered messages. Asynchronous communication with bounded delay can be easily modeled by using calendar as a global data structure. When a message is transmitted by a process, it is added to the

calendar as an event e_i to occur at time t_i, where t_i denotes the expected delivery time for the message. On receiving the message, the event is removed from the calendar. Thus at any state, the calendar C can be seen as a set of messages that have been sent but are yet to be received with corresponding expected delivery delays.

2.4 Limitation of Existing Formalisms

Timed automata is one of the most frequently used formalism for specifying real-time system designs. However as it turns out that for systems with asynchronous communication with bounded delays between components TA does not offer any efficient means of specification. Two possible choices have been considered in literature. First choice is to use state variables for encoding the behavior of asynchronous channels however without any explicit provision to capture message transmission delays. Second choice is to model each channel as a separate TA with delay as a state variable. However with relatively large number of components and high degree of connectivity among them, modeling channels in this way is difficult, and state space explosion becomes an unavoidable problem. UPPAAL [BDL04], which can model TA, has the same problem when it is used to model asynchronous communications with bounded delays - every channel has to be modeled as a separate TA capturing the message transmission delays.

On the other hand, although TTM and CTM are expressive enough to capture a range of behaviors associated with time triggered systems including asynchronous communication delays, they however have two specific design limitations:

- These models are not well suited for actual system design purpose since they describe the behavior of the combined system without (explicitly) specifying the design of the modular components.
- Absence of formally defined syntactic design models corresponding to these transitions systems would demand that additional correctness measures are put in place because for verification purposes actual designs models need to be (manually) interpreted and translated into these transition systems as per the underlying system dynamics and on discovering an error during verification, such errors need to comprehended by a designer, and subsequently, translated back into his design for a remedial action.

Keeping in view of such limitations in the existing specification formalisms, we will next define and elaborate using examples a new timeout based formalism, which can effectively overcome these barriers.

3 Formalization of Timeout and Calendar based Models

In [HMP92b] an abstract model of timed transition system was proposed which could represent a wide variety of behaviors of the timed execution of concurrent processes. In this section we adapt and extend the Timed Transition System (TTS) described therein to represent timeout and calendar based models. Further we describe their associated semantics in terms of state transition systems.

3.1 Timeout based Timed Transition Model

3.1.1 Syntax

A Timeout based Model (ToM) is represented as

$$P: \{\theta\}[P_1||P_2||\ldots||P_n].$$

Each process P_i is a sequential non-deterministic process having τ_i as its local timeout and $\mathcal{X}_i$ as a set of local timing variables. Local timing variables are used for determining the relative delay between events. A shared variable $\{t\}$ represents the global clock. The operator "||" denotes parallel composition. The formula θ, called the *data pre-condition* of P, restricts the initial values of variables in

$$\mathcal{U} = \{t\} \cup \mathcal{T} \cup \mathcal{X} \cup \mathit{Var},$$

where the set of all timeouts is $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, and $\mathcal{X} = \bigcup_i \mathcal{X}_i$. The set $\mathit{Var} = (G \cup L_1 \cup L_2 \cup \ldots \cup L_n)$ is the set of other state variables. The variables in G are globally shared among all the processes while L_i contains variables local to process P_i. Let f^{Var} be the set of computable functions on Var.

Each process P_i is represented using a *timeout transition diagram* (TTD), which is a finite directed graph with a set of nodes $\mathit{Loc}_i = \{l_0^i, l_1^i, \ldots, l_{m_i}^i\}$, called *locations*. The entry location is l_0^i. There are two

kinds of edges in the graph of a process P_i: *Timeout edges* and *Synchronous Communication edges*. Edge definitions involve an enabling condition or guard ρ, which is a boolean-valued function or a predicate.

Timeout Edges: A timeout edge $(l^i_j, \rho \Rightarrow \langle \tau_i := update_i, \gamma, f\rangle, l^i_k)$ in the graph of the process P_i is represented as

$$l^i_j \xrightarrow{\rho \Rightarrow \langle \tau_i := update_i, \gamma, f\rangle} l^i_k,$$

where $update_i$ specifies the way timeout τ_i is to be updated on taking a transition on the edge when the guard ρ evaluates to `true`. $\gamma \subseteq \mathcal{X}_i$ specifies the local timing variables which capture value of the clock t while taking transition on the edge. This value may be used during future transitions while estimating relative delay w.r.t. this transition. $f \in f^{Var}$ manipulates the state variables in $G \cup L_i$.

$update_i$ is defined using the rule:

$$update_i = k_1 \mid k_2 \mid \infty \mid \max(\mathcal{M})$$

where $l + z \prec k_1 \prec' m + z'$ for $\prec, \prec' \in \{<, \leq\}$ and $k_2 \succ l + z$ for $\succ \in \{>, \geq\}$; $z, z' := t|w$ and $l, m \in \mathbb{N}$ are non negative integer constants specifying the lower and upper limits for a timeout increment interval[1], and $w \in \mathcal{X}_i$ is a local timing variable. The variable z makes such an interval relative to the occurrence of specific events. $\mathcal{M}$ is the set of all the integer constants that are used to define the upper limit of different timeouts for different processes in the system. $\max(\mathcal{M})$ returns the maximum of all the integers in $\mathcal{M}$.

Constraints on k_1, k_2 specify how new value of timeout τ_i should be determined based upon the current value of the clock t and/or w, which would have captured the value of t in some earlier transition. Setting a timeout to ∞ tends to capture the requirement of indefinite waiting for an external signal/event. The selection of the timeout value using $\max(\mathcal{M})$ is used to capture the situation where the next discrete transition of a process may happen at any time in the future, for example, the process may be in a sleeping mode and can wake up at any future point of time.

Synchronous Communication Edges: Rendezvous communication between a pair of processes (P_s, P_r) is represented by having an edge pair (e_s, e_r) s.t. $e_s \in P_s$ and $e_r \in P_r$ and

$$e_s : l^s_j \xrightarrow{\rho \Rightarrow \langle ch!m, \tau_s := update_s, \gamma, g\rangle} l^s_k$$

$$e_r : l^r_j \xrightarrow{True \Rightarrow \langle ch?\bar{m}, \tau_i := update_r, \gamma', h\rangle} l^r_k$$

where ch is the channel name, $m \in L_i$ is the message sent, and $\bar{m} \in L_r$ the message received, and $g, h \in f^{Var}$ are the computable functions.

3.1.2 Semantics

With a given ToM

$$P: \ \{\theta\}[P_1||P_2||\ldots||P_n]$$

we associate the following transition system $S_P = (\mathcal{V}, \Sigma, \Sigma_0, \Gamma)$, referred to as *timeout based clocked transition system* (TCTS) where,

1. $\mathcal{V} = \mathcal{U} \cup \{\pi_1, \ldots, \pi_n\}$. Each *control variable* π_i ranges over the set $Loc_i \cup \{\perp\}$. The value of π_i indicates the location of the control for the process P_i and it is $\perp$ (undefined) before the start of the process.

2. Σ is the set of states. Every state $\sigma \in \Sigma$ is an interpretation of $\mathcal{V}$, that is, it assigns values to clock variable t, every timeout variable in $\mathcal{T}$, timing variables in $\mathcal{X}$, state variables in Var, and control variables $\pi_1, \ldots, \pi_n$, in their respective domains. For $x \in \mathcal{V}$, let $\sigma(x)$ denote its value in state σ.

3. $\Sigma_0 \subseteq \Sigma$ is the set of initial states such that for every $\sigma_0 \in \Sigma_0$, θ is true in σ_0 and $\sigma_0(\pi_i) = \perp$ for each process P_i.

4. $\Gamma = \Gamma_e \cup \Gamma_+ \cup \Gamma_0 \cup \Gamma_{syn\_comm}$ is the set of transitions. Every transition $\nu \in \Gamma$ is a binary relation on Σ defined further as follows:

[1] This interval mimics the delay interval marking an edge in the original timed transition diagrams

Entry Transitions: Γ_e, the set of entry transitions contains an *entry transition* ν_e^i for every process P_i. In particular, $\forall \sigma_0 \in \Sigma_0$,

$$\nu_e^i \equiv (\sigma_0, \sigma') \in \Gamma_e \Leftrightarrow \begin{cases} 1. & \forall x \in \mathcal{U} : \ \sigma'(x) = \sigma_0(x) \\ 2. & \forall \tau \in \mathcal{T} : \ \sigma'(t) \leq \sigma'(\tau) \\ 3. & \sigma_0(\pi_i) = \perp \ \text{and} \ \sigma'(\pi_i) = l_0^i \end{cases}$$

Time Progress Transition: The first kind of edges $\nu_+ \in \Gamma_+$ are those where the global clock is increased to the minimum of all timeouts. In particular,

$$\nu_+ \equiv (\sigma, \sigma') \in \Gamma_+ \Leftrightarrow \begin{cases} 1. & \sigma(t) < \min\{\sigma(\mathcal{T})\} \\ 2. & \forall \tau \in \mathcal{T} : \ \sigma'(\tau) = \sigma(\tau) \\ 3. & \forall x \in \mathcal{X} : \ \sigma'(x) = \sigma(x) \\ 4. & \forall i : \ \sigma'(\pi_i) = \ \sigma(\pi_i) \\ 5. & \sigma'(t) = \min\{\sigma(\mathcal{T})\} \end{cases}$$

Timeout Increment Transition: For the second kind of edges $\nu_0^i \in \Gamma_0$ the global clock equals the minimum of timeouts. Also if an edge in the TTD for process P_i connects source location l_j^i to target location l_k^i and is labeled by the instruction $\rho \Rightarrow \langle \tau_i := update_i, \gamma, f \rangle$, then

$$\nu_0^i \equiv (\sigma, \sigma') \in \Gamma_0 \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \textbf{If } \sigma(\tau_i) = \sigma(t) \\ & \quad \textbf{then } \sigma'(\tau_i) = update_i > \sigma(\tau_i) \\ & \textbf{else } \sigma'(\tau_i) = \sigma(\tau_i) \\ 4. & \forall x \in \gamma : \ \sigma'(x) = \sigma(t) \ \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma : \ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in G \cup L_i : \sigma'(v) = f(\sigma(v)) \text{ and} \\ & \forall v \in Var \setminus (G \cup L_i) : \ \sigma'(v) = \sigma(v) \\ 6. & \sigma(\pi_i) = \ l_j^i \text{ and } \sigma'(\pi_i) = l_k^i \end{cases}$$

If $update_i = k_1$ s.t. $l + z \prec k_1 \prec m + z'$, then $update_i$ arbitrarily selects a value δ such that $[l + \sigma(z) \prec \delta \prec m + \sigma(z')] \wedge [\delta > \sigma(\tau_i)]$ and returns δ. If $update_i = k_2$ s.t. $k_2 \succ l + z$, then $update_i$ arbitrarily selects a value δ such that $[\delta \succ l + \sigma(z)] \wedge [\delta > \sigma(\tau_i)]$ and returns δ. If $update_i = \infty$, $update_i$ returns the largest possible constant defined as per the design of the system. If $update_i = \max(\mathcal{M})$, $update_i$ nondeterministically selects any integer δ in $[0, M + 1]$, where M is the maximum of all the integers in $\mathcal{M}$ returned by $\max(\mathcal{M})$. The local timing variables in $\gamma \subseteq \mathcal{X}_i$ for process P_i are assigned the current value of global clock on timeout increment transition, while the other local timing variables in the system retain their old values before this transition. The variables in γ are thus used to capture the delay between two events.

Synchronous Communication: For a pair of processes P_s, P_r having synchronous communication edges (e_s, e_r) as defined before, $\nu_{syn\_comm}^{sr} \in \Gamma_{syn\_comm}$ exists such that:

$$\nu_{syn\_comm}^{sr} \equiv (\sigma, \sigma') \in \Gamma_{syn\_comm} \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \sigma'(\tau_s) = update_s > \sigma(\tau_s) \text{ and} \\ & \sigma'(\tau_r) = update_r > \sigma(\tau_r) \\ 4. & \forall x \in (\gamma \cup \gamma') : \ \sigma'(x) = \sigma(t) \ \text{ and} \\ & \forall x \in \mathcal{X} \setminus (\gamma \cup \gamma') : \ \sigma'(x) = \sigma(x) \\ 5. & \sigma'(\bar{m}) = \sigma(m) \\ 6. & \forall v \in G \cup L_s : \sigma'(v) = g(\sigma(v)) \\ & \forall v \in G \cup L_r : \sigma'(v) = h(\sigma(v)) \\ & \forall v \in Var \setminus (G \cup L_s \cup L_r) : \ \sigma'(v) = \sigma(v) \\ 7. & \sigma(\pi_s) = \ l_j^s, \sigma(\pi_r) = \ l_j^r \text{ and} \\ & \sigma'(\pi_s) = l_k^s, \sigma'(\pi_r) = \ l_k^r \end{cases}$$

This semantic model defines the set of possible computations of the ToM P as a (possibly infinite) set of state sequences $\xi : \ \sigma_0 \rightarrow \sigma_1 \rightarrow \ldots$, which starts with some initial state σ_0 in Σ_0 and follows with consecutive edges in Γ, *i.e.*, $\forall i.(\sigma_i, \sigma_{i+1}) \in \Gamma$. Let $[\![S_P]\!]$ be the set of all these computations of a ToM P as defined by its TCTS S_P.

3.2 Calendar Based Timed Transition Model

3.2.1 Syntax

Next we capture bounded message transfer delay associated with an asynchronous communication. Towards that the ToM is extended with a calendar data structure. A calendar is a linear list of bounded size, where each element of the list contains the following information: message, sender_id, receiver_id, and expected_delivery_time. Assuming $\mathcal{C}$ to denote the calendar array, a globally shared object, we set

$$\mathcal{U} = \{t\} \cup \mathcal{T} \cup \mathcal{X} \cup \mathit{Var} \cup \mathcal{C}$$

Sending a message in a TTD of process P_i is represented using the following edge:

$$l^i_j \xrightarrow{\rho \Rightarrow \langle send(m,i,\Omega), \tau_i := update_i, \gamma, f\rangle} l^i_k,$$

where $\Omega \subseteq R \times \Lambda$, $R \subseteq \{1, 2, \ldots n\}$ is the index set for the processes and Λ is the set of expected message delays. $send(\ldots)$ specifies that a message m is to be sent to each of the processes P_r with expected delivery time of λ_r where $(r, \lambda_r) \in \Omega$. On taking a transition on this edge an entry $\{$ m, i, r, $\lambda_r\}$ is added to $\mathcal{C}$ for each $(r, \lambda_r) \in \Omega$.

Receiving of the corresponding message is represented in the TTD for each of the processes $P_r, \forall\, r \in R$ using the following edge:

$$l^r_j \xrightarrow{True \Rightarrow \langle receive(m,i,r), \tau_r := update_r, \gamma, g\rangle} l^r_k,$$

where $receive(\ldots)$ specifies that a message m sent by process P_i is to be received by the process P_r. On taking a transition on this edge, the entry $\{$m, i, r, $\lambda_r\}$ is deleted from $\mathcal{C}$.

3.2.2 Semantics

Given a calendar $\mathcal{C}$, we assume that the set of delays for all undelivered messages at any state σ can be found using the function

$$\Delta : \sigma(\mathcal{C}) \to 2^{\mathbb{N}}$$

Again $\Gamma = \Gamma_e \cup \Gamma_+ \cup \Gamma_0 \cup \Gamma_{syn\_comm} \cup \Gamma_{asyn\_comm}$ is the set of transitions in the *calendar based clocked transition system* (CCTS). Both Γ_e (set of Entry Transition) and Γ_{syn_comm} (Synchronous Communication) are same as in TCTS defined earlier. The definitions for the edges in Time Progress Transition (Γ_+) and those for Timeout Increment Transition (Γ_0) are modified using calendar $\mathcal{C}$ as follows:

Time Progress Transition: The first kind of edges ν_+ are those where the global clock is increased to the minimum of all the timeouts and message delays. In particular,

$$\nu_+ \equiv (\sigma, \sigma') \in \Gamma_+ \Leftrightarrow \begin{cases} 1. & \sigma(t) < \min\{\sigma(\mathcal{T}) \cup \Delta(\sigma(\mathcal{C}))\} \\ 2. & \forall \tau \in \mathcal{T} :\ \sigma'(\tau) = \sigma(\tau) \\ 3. & \forall x \in \mathcal{X} \cup \mathit{Var} :\ \sigma'(x) = \sigma(x) \\ 4. & \forall i :\ \sigma'(\pi_i) = \ \sigma(\pi_i) \\ 5. & \sigma'(t) = \min\{\sigma(\mathcal{T}) \cup \Delta(\sigma(\mathcal{C}))\} \end{cases}$$

Timeout Increment Transition: For the second kind of edges ν^i_0 where global clock equals the minimum of all the timeouts and message delays, we have: if an edge in the TTD of process P_i connects source location l^i_j to target location l^i_k and is labeled by the instruction $\rho \Rightarrow \langle \tau_i := update_i, \gamma, f\rangle$, then

$$\nu^i_0 \equiv (\sigma, \sigma') \in \Gamma_0 \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \textbf{If } [\sigma(t) = \min\{\sigma(\mathcal{T})\}] \wedge [\sigma(\tau_i) = \sigma(t)] \\ & \quad \textbf{then } \sigma'(\tau_i) = update_i > \sigma(\tau_i) \\ & \textbf{else } \sigma'(\tau_i) = \sigma(\tau_i) \\ 4. & \forall x \in \gamma :\ \sigma'(x) = \sigma(t) \ \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma :\ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in G \cup L_i : \sigma'(v) = f(\sigma(v)) \ \text{ and} \\ & \forall v \in \mathit{Var} \setminus (G \cup L_i) :\ \sigma'(v) = \sigma(v) \\ 6. & \sigma(\pi_i) = \ l^i_j \text{ and } \sigma'(\pi_i) = l^i_k \end{cases}$$

We additionally define new transitions corresponding to *send*() and *receive*() to capture asynchronous communication:

Send Transition: If there is an edge in process P_i, which connects source location l^i_j to target location l^i_k and is labeled by the instruction $\rho \Rightarrow \langle send(m, i, \Omega), \tau_i := update_i, \gamma, f\rangle$, then we have the corresponding edge $\nu^i_{send} \in \Gamma_{asyn\_comm}$, which adds $|\Omega|$ cells to the calendar array $\mathcal{C}$:

$$\nu^i_{send} \equiv (\sigma, \sigma') \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \sigma'(\tau_i) = update_i > \sigma(\tau_i) \\ 4. & \forall x \in \gamma : \ \sigma'(x) = \sigma(t) \ \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma : \ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in G \cup L_i : \sigma'(v) = f(\sigma(v)) \ \text{ and} \\ & \forall v \in Var \setminus (G \cup L_i) : \ \sigma'(v) = \sigma(v) \\ 6. & \forall (r, \lambda_r) \in \Omega : \sigma'(\mathcal{C}) := \sigma(\mathcal{C}) \cup \{\mathsf{m}, \mathsf{i}, \mathsf{r}, \lambda_r\} \\ 7. & \sigma(\pi_i) = \ l^i_j \text{ and } \sigma'(\pi_i) = l^i_k \end{cases}$$

Receive Transition: If there is an edge in process P_r, which connects source location l^r_j to target location l^r_k and is labeled by the instruction $True \Rightarrow \langle receive(m, i, r), \tau_r := update_r, \gamma, g\rangle$, then we have the corresponding edge $\nu^r_{receive} \in \Gamma_{asyn\_comm}$, which deletes the entry $\{m, i, r, \lambda_r\}$ from the calendar array $\mathcal{C}$ when the clock t reaches λ_r:

$$\nu^r_{receive} \equiv (\sigma, \sigma') \Leftrightarrow \begin{cases} 1. & \exists \{m, i, r, \lambda_r\} \in \sigma(\mathcal{C}) \text{ s.t. } \sigma(t) = \lambda_r \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \sigma'(\tau_r) = update_r > \sigma(\tau_r) \\ 4. & \forall x \in \gamma : \ \sigma'(x) = \sigma(t) \ \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma : \ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in G \cup L_r : \sigma'(v) = g(\sigma(v)) \ \text{ and} \\ & \forall v \in Var \setminus (G \cup L_r) : \ \sigma'(v) = \sigma(v) \\ 6. & \sigma'(\mathcal{C}) := \sigma(\mathcal{C}) \setminus \{m, i, r, \lambda_r\} \\ 7. & \sigma(\pi_r) = \ l^r_j \text{ and } \sigma'(\pi_r) = l^r_k \end{cases}$$

Similar to the case of TCTS, this semantic model also defines the set of possible computations of the calendar based ToM as a (possibly infinite) set of state sequences starting with some initial state in Σ_0 and following consecutive edges in Γ. Let $[S_P]$ be the set of all these computations of a calendar based ToM P as defined by its CCTS S_P.

Models for Time: It remained unspecified as to what would be the underlying model of time for clock, timeouts etc that appear in the definitions of TCTS and CCTS. There are two natural choices for time, the set of non-negative integers $\mathbb{N}$ (discrete time) or the set of non-negative reals $\mathbb{R}$ (dense time). Given the model of time as *TIME*, let $[S_P]_{TIME}$ be the set of all the computations of a ToM (or calendar based ToM) P as defined by its TCTS (or CCTS) S_P.

When we consider that the underlying model of time as $\mathbb{R}$, we need to add the following non-zenoness condition to ensure effective time progress in the model. *There must not be infinitely many time progress (or timeout increment) transitions effective within a finite interval.* Formally,

nonzenoness:

$$\forall \xi : \ \sigma_0 \to \sigma_1 \to \ldots \in [S_P]_{\mathbb{R}}. \forall \delta \in \mathbb{R}. \exists i \geq 0. \sigma_i(t) > \delta$$

3.3 Parametric Processes

We consider the case of finite family of processes specified in a parametric way. A completely parametric process family would be specified as

$$\{\theta\}[\{P(i)\}_{i=1}^{i=N}]$$

where $N \geq 1$ is some finite positive integer and $\theta = \theta_1 \wedge \ldots \wedge \theta_N$ such that θ_i $(1 \leq i \leq N)$ initializes the variables for the i^{th} copy of the process. Process $P(i)$ could be a TTD or a calender based TTD.

The semantic interpretation of such parametrically specified process family is given by first flattening the specification as

$$\{\theta\}[P(1)|| \ldots ||P(N)]$$

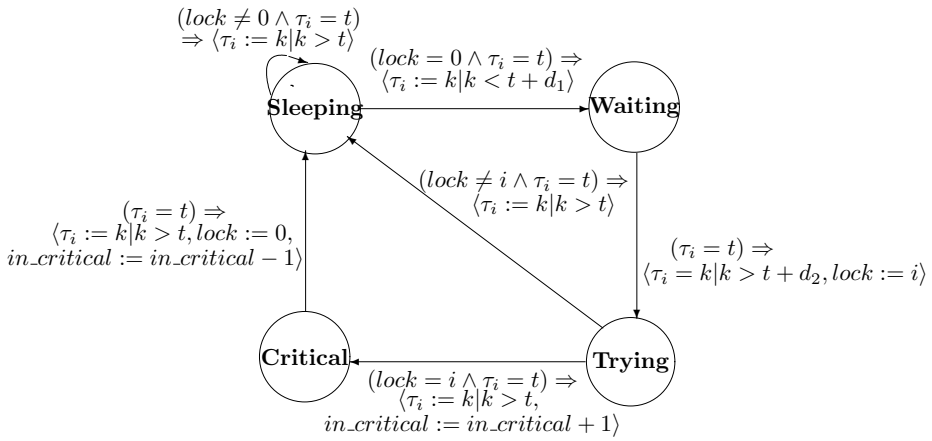

Figure 2: TTD for the i^{th} process in the Fischer's Protocol

and then applying the semantics presented before as per the case of $P(i)$ being a TTD or a calendar based TTD.

Such parametric specification can be generalized to a homogeneous set of process families as

$$\{\theta\}[\{P(i_1)\}_{i_1=1}^{i_1=N_1} || \ldots || \{P(i_l)\}_{i_l=1}^{i_l=N_l}]$$

where $N_1, \ldots N_l$ are some finite positive integers and $\theta = \theta_{\mathbf{1}} \wedge \ldots \wedge \theta_{\mathbf{l}}$ such that $\theta_{\mathbf{i}} = \theta_{i1} \wedge \ldots \wedge \theta_{iN_i}$ initializes the variables for the i^{th} process family. The term homogeneous arises because processes in all the process families should uniformly be either TTDs or calender based TTDs. We do not consider the case of hetrogeneous set of process families, where processes across different process families might be different. Similar to the case of a single parametric process family, the generalized process family can be interpreted by flattening the process specification.

4 Examples

Following two examples would illustrate the expressiveness and effectiveness of the proposed timeout and calendar based modeling framework.

4.1 Fisher's Mutual Exclusion Protocol

Fischer's protocol is a well studied protocol to ensure mutual exclusion among real time concurrent processes. Let there be n processes $P_1, \ldots, P_n$ trying to access shared resources in a real-time fashion to be discussed later. A process P_i is initially idle (*Sleeping* state), but at any time, may begin executing the protocol provided the value of a global variable *lock* is 0 and then move to *Wait* state. There it can wait up to maximum of d_1 time units before assigning the value i to *lock* and moving to *Trying* state. It may enter the *Critical* section after a delay of at least of d_2 time units provided the value of *lock* is still i. Otherwise it has to move to *Sleeping* state. Upon leaving the *Critical* section, it re-initializes *lock* to 0. There is another global variable, *in_critical*, used to keep count of the number of processes in the critical section. The auto-increment (auto-decrement) of the variable is done before a process enters the *Critical* section (leaves the *Critical* section). Mutual exclusion is ensured if $d_1 < d_2$. The timeout-based TTD of the i^{th} process P_i executing Fischer's protocol is shown in Figure 2.

4.2 TTA Startup Algorithm

The TTA startup algorithm can be formalized using the calendar based model described above. This algorithm executes on a logical bus meant for safety-critical application in both automotive and aerospace industries. In a normal operation, N processors or nodes share a TTA bus using a TDMA schedule. The goal of the startup algorithm is to bring the system from the power-up state, in which the processors are not synchronized, to the normal operation mode in which all processors are synchronized and follow the same TDMA schedule.

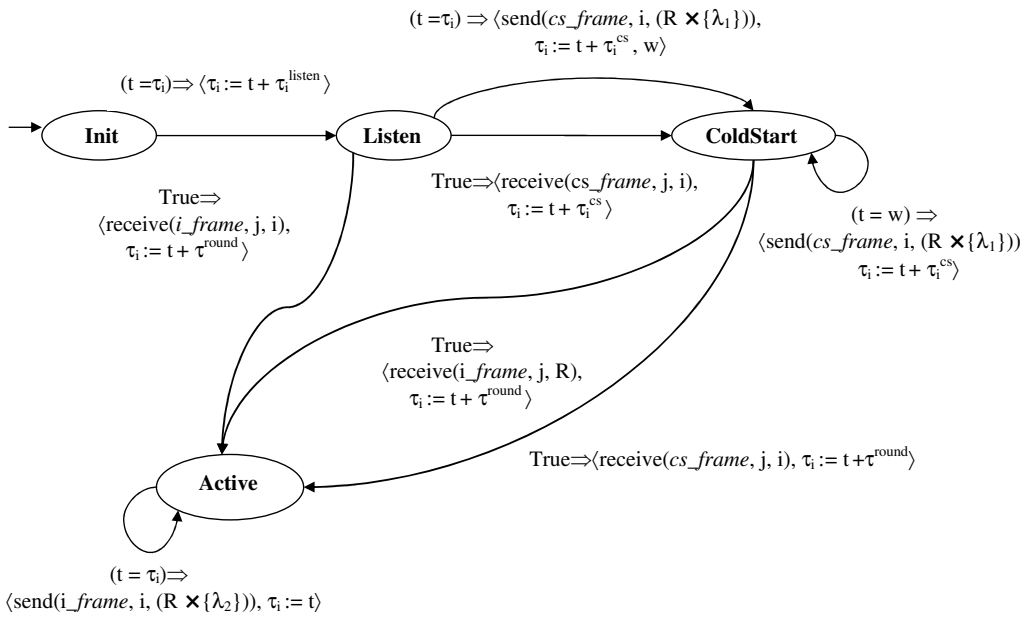

Figure 3: Calendar-based TTD for the i^{th} node in TTA Startup algorithm

In TTA startup algorithm each node $i \in \{1 \dots N\}$ has two unique timeout parameters, τ_i^{listen} and τ_i^{cs}, for *listen* and *coldstart* states respectively. These are defined as follows:

$$\tau_i^{listen} = 2\tau^{round} + \tau_i^{startup},$$
$$\tau_i^{cs} = \tau^{round} + \tau_i^{startup}$$

where τ^{round} represents the TDMA round duration and $\tau_i^{startup}$ denotes the duration between the start of a TDMA cycle and the time when the slot for node i starts. If τ denotes the duration of each slot then

$$\tau^{round} = N\tau, \quad \tau_i^{startup} = (i-1)\tau$$

When a node is powered-on, it performs some internal initialization, and transits to the *Listen* state. In this state it listens for the unique duration τ_i^{listen} to determine if there is a synchronous set of nodes communicating on the medium. The nodes which are in the *Active* state are already synchronized, and periodically transmit i-frames that carry the TDMA cycle structure. If a node in the *Listen* state receives such an i-frame, it adjusts its state to the frame contents and is thus synchronized with the set of already synchronous nodes. If the above does not happen, there are two possibilities. Each node listens for a cold-start message (cs-frame) from another node indicating the beginning of the cold-start sequence; cs-frames are similar to i-frames but carry a protocol state suggested by the sending node. When a node completes the reception of a cs-frame, it enters the *Coldstart* state and resets its local clock to δ_{cs} (that is the transmission duration of the cs-frame). Thus, all nodes that received the cs-frame have synchronized local clocks (within system tolerances, including the propagation delay). Each node that receives neither an i-frame nor a cs-frame during the *Listen* phase enters the *Coldstart* state on its listen timeout, resets its local clock to 0 and broadcasts a cs-frame. Thus, after the transmission of the cs-frame (δ_{cs} later), the local clock of the sending node is also synchronized to the local clocks of the set of receiving nodes.

Each node in the *coldstart* state waits for reception of another cs-frame or i-frame until its local clock reaches the value of its individual cold-start timeout. If it receives such a frame it synchronizes on its contents and enters the *Active* state; if not, it resets its local clock and again broadcasts a cs-frame. No further collision can occur at this point, because cold-start timeouts have a strict order and that is why no two nodes that caused a collision can collide again. The listen timeout of any node is greater than coldstart timeout of any node. No node which has come in the *Listen* state after the collision cannot move to the *Coldstart* state before the collision is resolved. For further details of startup protocol, we refer the reader to [StP02].

The calendar based TTD of the i^{th} node is depicted in Figure 3. In TTA startup algorithm, all the communications are asynchronous and hence, message delivery delays, which are finite and specified by

the designer have to be taken into account for correct operation of the protocol. The timeouts τ_i^{listen} and τ_i^{cs} represent how much time a node spends in *Listen* state and *Coldstart* state respectively, if no external signal is received. The timeout τ^{round} denotes the time a node spends in *Active* state before sending its next massage. $R_i = \{1, \ldots, N\} \setminus \{i\}$ represents the set of nodes except the sender i that are required to receive the message in the network. We use λ_i's to denote the message delivery time for the corresponding send events. In TTA, message delivery times for all the receivers are considered to be the same, and that is why we have considered a single variable λ_i to represent that delay.

5 Verification Results for Digitization

In literature the verification problem for real-time systems assumes two descriptions of real-time behavior, implementation I and specification S, and poses the question whether I implements/satisfies S. The implementation language $\mathcal{L}_I$ describes systems and behavior over time while the specification language $\mathcal{L}_S$ describes the timing requirements of the system. The verification obligation involves presenting algorithms and/or proof rules that facilitate a formal argument that a particular implementation meets the requirement of a particular system under some particular assumption of semantics of computation and time. Assuming C and T to be mathematical models of computation and time respectively, the real-time verification problem parameterized by $(C, T, \mathcal{L}_I, \mathcal{L}_S)$ states: does the implementation of the system I, given as an expression of $\mathcal{L}_I$ meet the specification ϕ given as an expression of $\mathcal{L}_S$, with respect to the semantical assumption (C, T), written as

$$I \models^{?}_{(C,T)} \phi$$

In particular, we would consider two important instances of the real-time verification problem - one with an *integral* model of time and one with a *dense* model of time. In the following, we assume TTS as the implementation language and linear time temporal logic (LTL) as the specification formalism.

5.1 Timed Sequences

We shall adopt discrete trace model (using the terminology from [HMP92a, Bos99]) as a mathematical model of computation. By *discrete trace* model one can capture the behavior of a system as an infinite sequence of snapshots of the global system state at certain times. We assume our time domain $TIME$ has a total ordering $\leq$ defined on it. We define an *observation* to be a pair (σ_i, T_i), where σ_i is a state and $T_i \in TIME$. A *timed state sequence* $\eta = (\sigma, T)$ is an infinite sequence $\eta : (\sigma_0, T_0) \rightarrow (\sigma_1, T_1) \rightarrow (\sigma_2, T_2) \rightarrow \cdots$ of observations[2]. Further, the infinite sequence $T_i \in T$ of time stamps in η satisfy (i) *monotonicity*: $T_i \leq T_{i+1}$ for all $i \geq 0$, and (ii) *progress:* time progresses, for all $T \in TIME$, $T_i \geq T$ for some $i \geq 0$.

Now onwards, we shall work with dense-time models when $TIME = \mathbb{R}$ and integral-time models when $TIME = \mathbb{N}$. A timed state sequence under dense-time model will be referred to as *precisely timed* and under integral-time model as *digitally timed*.

Let us denote the set of all timed state sequences over the $TIME$ domain as TSS_{TIME}. A *real-time* property is a subset of TSS_{TIME}. Every real-time system S defines a real-time property, denoted as $[\![S]\!]$, which is the set of all timed state sequences of S. Also, every real-time specification ϕ defines a real-time property $[\![\phi]\!]$, the set of real-time sequences that satisfy ϕ.

Now let us formulate the real-time verification problem. We say a real-time system S satisfies the specification ϕ, written as

$$S \models_{TIME} \phi$$

if and only if

$$[\![S]\!]_{TIME} \subseteq [\![\phi]\!]_{TIME}$$

Consider a dense-time property $\Pi_{\mathbb{R}} \subseteq TSS_{\mathbb{R}}$, a set of of timed state sequences over $\mathbb{R}$. Its *clock-independent semantics* $\mathbb{N}(\Pi_{\mathbb{R}})$ is the subset of digitally timed state sequences in $\Pi_{\mathbb{R}}$, *i.e.,* $\mathbb{N}(\Pi_{\mathbb{R}}) = \Pi_{\mathbb{R}} \cap TSS_{\mathbb{N}}$. In [HMP92a], it is shown that clock-independent semantics is not very adequate for reasoning about dense time. As a remedy of this, another approximate semantics was introduced, which was called *digitization*.

[2]Note that any $\xi \in [\![S_P]\!]$ (previously defined) essentially defines a timed state sequence. This is because, states in ξ have implicit representation for time stamps as $\sigma_0(t), \sigma_1(t), \ldots$, which are otherwise explicitly present in the definition of η as $T_0, T_1, \ldots$

The following definitions will be useful for our subsequent discussions. For any timed state sequence $\eta = (\sigma, T)$, we introduce it untime operation η^- as its state component σ. Also, $\eta^i = (\sigma^i, T^i)$, for $i \geq 0$, denotes the timed state sequence that results from η by deleting the first i observations (note, $\eta^0 = \eta$).

5.2 Digitization

Given $x \in \mathbb{R}$ and $\epsilon \in (0, 1]$, we define $[x]_\epsilon = \lfloor x \rfloor$ if $x \leq \lfloor x \rfloor + \epsilon$, otherwise $[x]_\epsilon = \lceil x \rceil$[3]. Given a precisely timed sequence $\eta = (\sigma, T)$ and $\epsilon \in (0, 1]$, we define the *ϵ-digitization* $[\eta]_\epsilon = (\sigma, [T]_\epsilon)$ of η be the digitally timed sequence

$$(\sigma_0, [T_0]_\epsilon) \rightarrow (\sigma_1, [T_1]_\epsilon) \rightarrow \cdots,$$

For any dense-time property Π (a set of timed sequences over dense time) let

$$[\Pi] = \{[\eta]_\epsilon \mid \eta \in \Pi \text{ and } \epsilon \in (0, 1]\},$$

which is a digitization of Π. We write $[\eta]$ instead of $[\{\eta\}]$.

We state some concepts from [HMP92a]. Let Π be a dense-time property. Π is *closed under digitization* iff for all $\eta \in TSS_{\mathbb{R}}, \eta \in \Pi$ implies $[\eta] \subseteq \Pi$. Π is *closed under inverse digitization* iff $[\eta] \subseteq \Pi$ implies $\eta \in \Pi$, for all $\eta \in TSS_{\mathbb{R}}$. Finally, Π is *digitizable* iff it is closed under both digitization and inverse digitization, *i.e.,* $\eta \in \Pi$ iff $[\eta] \subseteq \Pi$ for all $\eta \in TSS_{\mathbb{R}}$. We state the following important result (see [HMP92a]).

Fact 5.1 *Assume a real-time system S whose dense-time semantics $[\![S]\!]_{\mathbb{R}}$ is closed under digitization, and a specification ϕ whose dense-time semantics $\phi_{\mathbb{R}}$ is closed under inverse digitization. Then in order to prove $S \models_{\mathbb{R}} \phi$ it suffices to check if $S \models_{\mathbb{N}} \phi$.*

A dense-time property Π is said to be *qualitative* if $\eta \in \Pi$ implies $\eta' \in \Pi$ for all precisely timed sequences η and η' with identical state components (*i.e.,* $\eta^- = \eta'^-$).

Fact 5.2 *[HMP92a] Every qualitative property is digitizable.*

5.3 Digitization of Timeout and Calendar based Transition Systems

Recall a TCTS is $S = (\mathcal{V}, \Sigma, \Sigma_0, \Gamma)$ (we drop the subscript P because we assume the ToM P is implicit) where $\mathcal{V}$ is a set of variables, Σ a set of states, $\Sigma_0 \subseteq \Sigma$ a set of initial states and Γ a set of transitions. We would like to show that the computations for this transition system are digitizable. Our approach follows [Bos99].

A *run* of S over a timed state sequence $\eta : (\sigma_0, T_0) \rightarrow (\sigma_1, T_1) \rightarrow \cdots$ is a sequence of pairs of S of the form $\zeta : (\sigma_0, \nu_0) \stackrel{T_1}{\rightarrow} (\sigma_1, \nu_1) \stackrel{T_2}{\rightarrow} \cdots$ where σ_i denotes the state and ν_i the mapping of variables in $\mathcal{U}$ in state σ_i and further, it satisfies the following conditions:

1. (initiation:) $\sigma_0 \in \Sigma_0$ and $\nu_0(t) = T_0, \ \ \forall i \geq 0. \nu_0(\pi_i) = \perp, \ \ t \in \mathcal{V}, \pi_i \in \mathcal{V}$.
2. (consecution:) for $i \geq 1$ there is an edge $(\sigma_{i-1}, \sigma_i) \in \Gamma = (\Gamma_e \cup \Gamma_+ \cup \Gamma_0 \cup \Gamma_{syn\_comm})$ such that the following hold:
   - if $(\sigma_0, \sigma_1) \in \Gamma_e$ then $T_0 = \nu_0(t) \leq \nu_1(t) = T_1$ and $\forall \tau \in \mathcal{T}. \sigma_1(\tau) \geq T_1$.
   - if $(\sigma_{i-1}, \sigma_i) \in \Gamma_+$ then $T_{i-1} = \nu_{i-1}(t) < \min\{\sigma_{i-1}(\mathcal{T})\} = \nu_i(t) = T_i$.
   - if $(\sigma_{i-1}, \sigma_i) \in \Gamma_0$ then $T_{i-1} = \nu_{i-1}(t) = \nu_i(t) = T_i$.
   - if $(\sigma_{i-1}, \sigma_i) \in \Gamma_{syn\_comm}$ then $T_{i-1} = \nu_{i-1}(t) = \nu_i(t) = T_i$.
3. (time progress:) for any real number T there exists an $i \geq 0$ such that $T_i > T$.

We say that $\eta \in TSS_{\text{TIME}}$ is *time-consistent* (for S) if S has a run over it. In the sequel we consider only time-consistent behaviors $\eta \in [\![S]\!]_{\text{TIME}}$ of S, *i.e.,* $\eta \in [\![S]\!]_{\text{TIME}}$ iff there is run over η. If TIME $= \mathbb{N}$ then we get integral behavior of TCTS. Now it is obvious that time at state $j \geq 1$ in a given run, is given by $\nu_j(t) = T_j$. We define ϵ-digitization of the mapping ν_j for any variable $x \in \mathcal{U} \subseteq \mathcal{V}$ as $\langle \nu_j(x) \rangle_\epsilon = [\nu(x)]_\epsilon$.

Given a computation $\zeta : (\sigma_0, \nu_0) \stackrel{T_1}{\rightarrow} (\sigma_1, \nu_1) \stackrel{T_2}{\rightarrow} \cdots$ its ϵ-digitization is the computation $[\zeta]_\epsilon : (\sigma_0, \langle \nu_0 \rangle)_\epsilon) \stackrel{[T_1]_\epsilon}{\rightarrow} (\sigma_1, \langle \nu_1 \rangle)_\epsilon \stackrel{[T_2]_\epsilon}{\rightarrow} \cdots$, where $\langle \nu_j \rangle_\epsilon$ for $j \geq 1$ are defined above, and $\langle \nu_0(t) \rangle_\epsilon = [T_0]_\epsilon$.

[3] where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are the floor and ceiling rounding operations on real numbers respectively

Now we need to analyze the extent to which the set of dense-time computations of a TCTS are closed under digitization. Suppose $\zeta : (\sigma_0, \nu_0) \stackrel{T_1}{\rightarrow} (\sigma_1, \nu_1) \stackrel{T_2}{\rightarrow} \cdots$ is a run of S over η. For digitization, $[\zeta]_\epsilon$ would be a run of S over $[\eta]_\epsilon$. We have $\langle \nu_0(t) \rangle_\epsilon = [T_0]_\epsilon$. Observe if $T_{i-1} = T_i$ then $[T_{i-1}]_\epsilon = [T_i]_\epsilon$. When $T_{i-1} < T_i$, except for the case of $0 < (T_i - T_{i-1}) < 1$, we have $[T_{i-1}]_\epsilon < [T_i]_\epsilon$. So, if there is an edge $(\sigma_{i-1}, \sigma_i) \in \Gamma$ and $(T_i = T_{i-1}) \vee (T_i \geq T_{i-1} + 1)$, there would be an edge $(\langle \sigma_{i-1} \rangle_\epsilon, \langle \sigma_i \rangle_\epsilon)$ in Γ under $[\zeta]_\epsilon$. Also we can ensure time progress for $[\zeta]_\epsilon$. Hence:

Fact 5.3 *The set of dense-time computations of a TCTS are closed under digitization if and only if all timeout increments are at least* 1 *time unit.*

The result above indicates a precise characterization for the digitization for a TCTS. All timeout increments in $(0, 1)$ result into a TCTS, which are not closed under digitization and therefore cannot be model checked for all LTL properties under discrete time dynamics.

A similar argument can be used to show that the dense computations of a (digitizable) calendar based clocked transition system (CCTS) are also closed under digitization.

5.4 Linear Temporal Logic

Let us briefly describe propositional linear temporal logic [Pnu77], more popularly known as LTL. The vocabulary of LTL consist of a set $\mathcal{P}$ of atomic propositions. The formulas of LTL are built using boolean connectives, next operator $\bigcirc$ and *until* operator $\mathcal{U}$ as follows:

$$\phi \ ::= \ p \,|\, \neg\phi \,|\, \phi_1 \wedge \phi_2 \,|\, \bigcirc \phi \,|\, \phi_1 \mathcal{U} \phi_2, \quad p \in \mathcal{P}$$

The other temporal operators can be introduced as abbreviations, *e.g.*, $F\phi \ \widehat{=} \ True\, \mathcal{U}\, \phi, G\phi \ \widehat{=} \ \neg F \neg \phi$.

The formulas of LTL can be interpreted over timed state sequences whose states are from Σ such that each state in Σ gives rise to an interpretation for propositions in $\mathcal{P}$. Let $\eta = (\sigma, T)$ be a timed state sequence with $\sigma_i \in \Sigma$ for $i \geq 0$. The satisfaction relation $\eta \models \phi$ is defined inductively as follows:

$\eta \models p$	iff	$\sigma_0 \models p$;
$\eta \models \neg\phi$	iff	$\eta \not\models \phi$;
$\eta \models \phi_1 \wedge \phi_2$	iff	$\eta \models \phi_1$ and $\eta \models \phi_2$
$\eta \models \bigcirc\phi$	iff	$\eta^1 \models \phi$ and $T_1 \geq T_0$,
$\eta \models \phi_1 \mathcal{U} \phi_2$	iff	$\exists i \geq 0 . \exists \alpha \in \mathbb{N} . \eta^i \models \phi_2$, where
		$T_i \geq T_0 + \alpha$, and $\forall j . 0 \leq j < i . \eta^j \models \phi_1$.

For a LTL-formula ϕ, let the set $[\![\phi]\!]_{TIME} \subseteq TSS_{\text{TIME}}$ contain all timed state sequences η over the time domain $TIME$ such that $\eta \models \phi$. Thus, $[\![\phi]\!]_{\mathbb{R}}$ is the analog dense-time property for the formula ϕ. Note that for any specification ϕ expressed in LTL, $[\![\phi]\!]_{\mathbb{R}}$ is closed under inverse digitization. To see this consider two timed sequences η and η' with identical state components. Suppose $\eta \models \phi$, *i.e.*, $\eta \in [\![\phi]\!]_{\mathbb{R}}$. Now the proof is by induction on the structure of ϕ. At the induction stage, we only consider the case $\phi = \phi_1 \mathcal{U} \phi_2$. Now $\eta \models \phi_1 \mathcal{U} \phi_2$ iff for some $i \geq 0, \alpha \in \mathbb{N}, \eta^i \models \phi_2$, where $T_i \geq T_0 + \alpha$, and $\eta^j \models \phi_1$ for all $0 \leq j < i$. By induction hypothesis, we have $\eta'^i \models \phi_2$ and $\eta'^j \models \phi_1$. Since, $T'_i \geq T'_0$, there exists some $\alpha' \in \mathbb{N}$ such that $T'_i \geq T'_0 + \alpha'$. Therefore $\eta' \models \phi$ and hence $\eta' \in [\![\phi]\!]_{\mathbb{R}}$.

5.5 An Integral Verification Problem

We conclude this section with this important observation. Given a TCTS or CCTS S, corresponding to a timeout-based or a calender-based model and a specification formula ϕ in LTL we may check $S \models_{\mathbb{R}} \phi$ by verifying whether $S \models_{\mathbb{N}} \phi$. In the next section we shall try to further simplify this problem.

6 Clockless Modeling

A finite state model-checker like Spin [Hol93] uses finite state automata to model the behavior of concurrent processes in distributed systems. The combined execution of a system of asynchronous processes is described as a product of automata each of which models an individual process. The product automaton is finite if the number of processes, message channels, number of messages in a channel, and the range of values for various variables are finite in the automaton for each individual process.

Though timeout and calendar based models can be used to efficiently capture dense time semantics without using a continuously varying clock, it is difficult to use these models for finite state model checking, even though we have seen that in most of the cases the verification problem reduces to an integral one thanks to digitization. The difficulty arises from the fact that the value of the global clock t and the values of the timeout variables in $\mathcal{T}$ diverge and thus are not bounded by a finite domain. Unlike TA there is no provision of resetting the global clock or timeouts in these models, as a result of which the timeout and calendar based models cannot be directly used for finite state model checking.

We propose a finitary reduction technique, which is formalized in terms of clockless modeling and semantics in the next section. This technique effectively reduces the timeout and calendar based transition systems with discrete dynamics into finite state systems, which, in turn, can be expressed and model checked by finite state model checkers. The assumption of discrete time as the underlying model is particularly relevant to cases where we are left with integral verification problem exploiting digitization results.

From the semantics of the timeout based systems it is clear that to implement time progress transition, a special process is required to increase the global clock to the minimum of timeouts, when each of the timeout values is strictly greater than the current value of the clock. A process P_i waits until its timeout is equal to global clock, and when it is so, P_i takes the discrete transition and updates its own timeout according to the specified updation rule. We model this special process, which is responsible for time progress transition in such a way that it does not explicitly use the clock variable and prevents the timeout variables from growing infinitely. We call this process as *time_progress*.

The process *time_progress* is implemented as follows. When the global clock is less than all the timeouts no discrete transition is possible in the system. In such a situation, *time_progress* finds out the minimum of all the timeouts in $\mathcal{T}$ and scales down all these timeouts in $\mathcal{T}$ by this amount. In this way at least one of the timeouts becomes zero. The guards of the processes are defined in such a way that the processes wait until their timeouts become zero. When it happens the process updates its timeout and does other necessary jobs.

If *update* function always increments the timeouts by a finite value then it is guaranteed that the value of a timeout will always be in a finite domain. But in some cases it is possible that a timeout may take any value in the future. In those cases, the value of the timeout is taken as the largest possible value defined by the system. This approach can also be extended for the calendar based models as well.

The discussion above is formalized in terms of "clockless" modeling as below:

6.1 Timeout based Models: Clockless Modeling

6.1.1 Clockless Syntax

In order to capture the effect of finite state reduction in a timeout model, we restrict the set $\mathcal{U}$ and redefine $update_i$ as follows:

$$\mathcal{U} = \mathcal{T} \cup \mathcal{X} \cup Var.$$

$update_i^-$ is given by the following rule:

$$update_i^- = k_1 \mid k_2 \mid \infty \mid \max(\mathcal{M}),$$

where $l - z \prec k_1 \prec' m - z'$ for $\prec, \prec' \in \{<, \leq\}$ and $k_2 \succ l - z$ for $\succ \in \{>, \geq\}$; $z, z' := w|0$ and $l, m \in \mathbb{N}$ are non negative integer constants. For any $z \in U$ let $\sigma_i^-(z)$ stand for the value of the variable z in (clockless) state σ_i^-. Note that $update_i^-$ is different from the update function $update_i$ for clocked transition system in the sense that this one updates the timeouts in bounded domain.

6.1.2 Clockless Semantics

For clockless modeling of timeout based models we associate a transition system $S_P^- = (\mathcal{V}^-, \Sigma^-, \Sigma_0^-, \Gamma^-)$, where $\mathcal{V}^- = \mathcal{V} \setminus \{t\}$ is a set of variables, Σ^- a set of clockless states, $\Sigma_0^- \subseteq \Sigma^-$ initial clockless states (defined in an analogous manner as for clocked transition systems) and Γ^- a set of clockless transitions. We remark that given a timeout based model, the set of states Σ for clocked transition system and the set of states Σ^- for clockless transition system are exactly similar modulo the assignment of the global clock variable t. The same is true for initial states too. Note $\Gamma^- = \Gamma_e^- \cup \Gamma_+^- \cup \Gamma_0^- \cup \Gamma_{syn\_comm}^-$, while Γ_e^- is identical to Γ_e for clocked transitions, we shall only define Time Progress Transition Γ_+^-, Timeout Increment Transition Γ_0^-, and Synchronous Communication Transition $\Gamma_{syn\_comm}^-$ by modifying the same for the clocked timeout transition system as defined earlier.

Time Progress Transition: The edges ν_+ are redefined such that all the timeouts are decremented by the minimum of the current timeout values. In particular,

$$\nu_+ \equiv (\sigma^-, \sigma'^-) \in \Gamma_+^- \Leftrightarrow \begin{cases} 1. & \min\{\sigma^-(\mathcal{T})\} > 0 \\ 2. & \forall \tau \in \mathcal{T} : \ \sigma'^-(\tau) = \sigma^-(\tau) - \min\{\sigma^-(\mathcal{T})\} \\ 3. & \forall x \in \mathcal{X} \cup \mathit{Var} : \ \sigma'^-(x) = \sigma^-(x) \\ 4. & \forall i : \ \sigma'^-(\pi_i) = \ \sigma^-(\pi_i) \end{cases}$$

Timeout Increment Transition: For the edges ν_0^i, if there is an edge in the TTD for process P_i connecting source location l_j^i to target location l_k^i and is labeled by the instruction $\rho \Rightarrow \langle \mathit{update}_i^-, \gamma, f \rangle$, then

$$\nu_0^i \equiv (\sigma^-, \sigma'^-) \in \Gamma_0^- \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma^- \\ 2. & \textbf{If } \sigma^-(\tau_i) = 0 \textbf{ then} \\ & \quad \sigma'^-(\tau_i) = \mathit{update}_i^- > 0 \\ & \textbf{else } \sigma'^-(\tau_i) = \sigma^-(\tau_i) \\ 3. & \forall y \in \gamma : \ \sigma'^-(y) = \sigma'^-(\tau_i) + \sigma^-(y) \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma : \ \sigma'^-(x) = \sigma^-(x) \\ 4. & \forall v \in G \cup L_i : \sigma'^-(v) = f(\sigma^-(v)) \text{ and} \\ & \forall v \in \mathit{Var} \setminus (G \cup L_i) : \ \sigma'^-(v) = \sigma^-(v) \\ 5. & \sigma^-(\pi_i) = \ l_j^i \text{ and } \sigma'^-(\pi_i) = l_k^i \end{cases}$$

Observe that update_i^- is a slight modification of update_i. If $\mathit{update}_i^- = k_1$ s.t. $l - z \prec k_1 \prec m - z'$, then update_i^- arbitrarily selects a value δ such that $l - \sigma^-(z) \prec \delta \prec m - \sigma^-(z')$. If $\mathit{update}_i^- = k_2$ s.t. $k_2 \succ l - z$, then update_i^- arbitrarily selects a value δ such that $\delta \succ l - \sigma^-(z)$, else if $\mathit{update}_i^- = \infty$, then it selects the largest possible constant defined by the system and returns δ. If $\mathit{update}_i^- = \max(\mathcal{M})$, update_i^- nondeterministically selects any integer δ in $[0, M+1]$, where M is the maximum of all the integers in $\mathcal{M}$. Unlike the local timing variables appearing in γ in a (clocked) ToM, these timing variables incrementally capture the value of next timeout in a clockless ToM. An observant reader can see that the relative delay captured by these local timing variables between events are same in both those models.

Synchronous Communication For a pair of processes P_s, P_r having edges (e_s, e_r) :

$$\nu_{\mathit{syn\_comm}}^{sr} \equiv (\sigma^-, \sigma'^-) \in \Gamma_{\mathit{syn\_comm}}^- \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma^- \\ 2. & \sigma'^-(\tau_s) = \mathit{update}_s^- > \sigma^-(\tau_s) \\ & \sigma'^-(\tau_r) = \mathit{update}_r^- > \sigma^-(\tau_r) \\ 3. & \forall y \in (\gamma) : \ \sigma'^-(y) = \sigma'^-(\tau_s) + \sigma^-(y), \ \text{ and} \\ & \forall y' \in (\gamma') : \ \sigma'^-(y') = \sigma'^-(\tau_r) + \sigma^-(y') \ \text{ and} \\ & \forall x \in \mathcal{X} \setminus (\gamma \cup \gamma') : \ \sigma'^-(x) = \sigma^-(x) \\ 4. & \sigma'^-(\bar{m}) = \sigma^-(m) \\ 5. & \forall v \in G \cup L_s : \sigma'^-(v) = g(\sigma^-(v)), \ \text{ and} \\ & \forall v \in G \cup L_r : \sigma'^-(v) = h(\sigma^-(v)) \ \text{ and} \\ & \forall v \in \mathit{Var} \setminus (G \cup L_r \cup L_s) : \ \sigma'^-(v) = \sigma^-(v) \\ 6. & \sigma^-(\pi_s) = \ l_j^s, \sigma^-(\pi_r) = \ l_j^r \text{ and} \\ & \sigma'^-(\pi_s) = l_k^s, \sigma'^-(\pi_r) = \ l_k^r \end{cases}$$

6.2 Calendar based Models: Clockless Modeling

6.2.1 Clockless Syntax

Similar to the ToM, calendar based models can also be defined in a clockless manner. However we restrict the set $\mathcal{U}$ to,

$$\mathcal{U} = \mathcal{T} \cup \mathcal{X} \cup \mathit{Var} \cup \mathcal{C},$$

where update_i^- is defined using same rule as in the case of clockless ToM.

6.2.2 Clockless Semantics

Similar to the clockless ToM, we can define a transition system for clockless calendar based models. Here we need to modify the Time Progress, Timeout Increment, Send, and Receive Transitions as defined earlier for CCTS. Synchronous Communication transition is similar to the one for timeout based model

with clockless semantics.

Time Progress Transition: The first kind of edges ν_+ are redefined so that all the timeout and calendar delay entries are decremented by the minimum of all timeouts and the message delays in calendar. In particular,

$$\nu_+ \equiv (\sigma^-, \sigma'^-) \in \Gamma_+^- \Leftrightarrow \begin{cases} 1. & \min\{\sigma^-(\mathcal{T}) \cup \Delta(\sigma^-(\mathcal{C}))\} > 0 \\ 2. & \forall \tau \in \mathcal{T} : \ \sigma'^-(\tau) = \sigma^-(\tau) - \min\{\sigma^-(\mathcal{T}) \cup \Delta(\sigma^-(\mathcal{C}))\} \\ 3. & \forall \lambda \in \Delta(\sigma^-(\mathcal{C})) : \ \sigma'^-(\lambda) = \sigma^-(\lambda) - \min\{\sigma^-(\mathcal{T}) \cup \Delta(\sigma^-(\mathcal{C}))\} \\ 4. & \forall x \in \mathcal{X} \cup \mathit{Var} : \ \sigma'^-(x) = \sigma^-(x) \\ 5. & \forall i : \ \sigma'^-(\pi_i) = \ \sigma^-(\pi_i) \end{cases}$$

Timeout Increment Transition: For the second kind of edges ν_0^i, if there is an edge in process P_i connecting source location l_j^i to target location l_k^i and is labeled by the instruction $\rho \Rightarrow \langle \tau_i := \mathit{update}_i^-, \gamma, f \rangle$, then

$$\nu_0^i \equiv (\sigma^-, \sigma'^-) \in \Gamma_0^- \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma^- \\ 2. & \textbf{If } \min\{\sigma^-(\mathcal{T})\} = \sigma^-(\tau_i) = 0 \\ & \quad \textbf{then } \sigma'^-(\tau_i) = \mathit{update}_i^- > 0 \\ & \textbf{else } \sigma'^-(\tau_i) = \sigma^-(\tau_i) \\ 3. & \forall y \in \gamma : \ \sigma'^-(y) = \sigma'^-(\tau_i) + \sigma^-(y) \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma : \ \sigma'^-(x) = \sigma^-(x) \\ 4. & \forall v \in G \cup L_i : \sigma'^-(v) = f(\sigma^-(v)) \text{ and} \\ & \forall v \in \mathit{Var} \setminus (G \cup L_i) : \ \sigma'^-(v) = \sigma^-(v) \\ 5. & \sigma^-(\pi_i) = \ l_j^i \text{ and } \sigma'^-(\pi_i) = l_k^i \end{cases}$$

Send Transition: If there is an edge in process P_i, which connects source location l_j^i to target location l_k^i and is labeled by the instruction $\rho \Rightarrow \langle \mathit{send}(m, i, \Omega, \Lambda), \mathit{update}_i^-, \gamma, f \rangle$, then we have corresponding edge ν_{send}^i which adds $|\Omega|$ cells to the calendar array $\mathcal{C}$:

$$\nu_{send}^i \equiv (\sigma^-, \sigma'^-) \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma^- \\ 2. & \textbf{If } \min\{\sigma^-(\mathcal{T})\} = \sigma^-(\tau_i) = 0 \\ & \quad \textbf{then } \sigma'^-(\tau_i) = \mathit{update}_i^- > 0 \\ & \textbf{else } \sigma'^-(\tau_i) = \sigma^-(\tau_i) \\ 4. & \forall y \in \gamma : \ \sigma'^-(y) = \sigma'^-(\tau_i) + \sigma^-(y) \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma : \ \sigma'^-(x) = \sigma^-(x) \\ 5. & \forall v \in G \cup L_i : \sigma'^-(v) = f(\sigma^-(v)) \text{ and} \\ & \forall v \in \mathit{Var} \setminus (G \cup L_i) : \ \sigma'^-(v) = \sigma^-(v) \\ 6. & \forall (r, \lambda_r) \in \Omega : \sigma'^-(\mathcal{C}) := \sigma^-(\mathcal{C}) + \{\mathsf{m}, \mathsf{i}, \mathsf{r}, \lambda_r\} \\ 7. & \sigma^-(\pi_i) = \ l_j^i \text{ and } \sigma'^-(\pi_i) = l_k^i \end{cases}$$

Receive Transition: If there is an edge in process P_r, which connects source location l_j^r to target location l_k^r and is labeled by the instruction $\mathit{True} \Rightarrow \langle \mathit{receive}(m, i, r), \gamma, f \rangle$, then we have corresponding edge $\nu_{receive}^r$ which deletes the cell containing $\{m, i, r, \lambda_r\}$ from the calendar array $\mathcal{C}$:

$$\nu_{receive}^r \equiv (\sigma^-, \sigma'^-) \Leftrightarrow \begin{cases} 1. & \exists \{m, i, r, \lambda_r\} \in \sigma^-(\mathcal{C}) \text{ s.t. } \lambda_r = 0 \\ 2. & \sigma'^-(\tau_r) = \mathit{update}_r^- > 0 \\ 3. & \forall y \in \gamma : \ \sigma'^-(y) = \sigma'^-(\tau_i) + \sigma^-(y) \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma : \ \sigma'^-(x) = \sigma^-(x) \\ 4. & \forall v \in G \cup L_r : \sigma'^-(v) = f(\sigma^-(v)) \text{ and} \\ & \forall v \in \mathit{Var} \setminus (G \cup L_r) : \ \sigma'^-(v) = \sigma^-(v) \\ 5. & \sigma'^-(\mathcal{C}) := \sigma^-(\mathcal{C}) \setminus \{m, i, r, \lambda_r\} \\ 6. & \sigma^-(\pi_r) = \ l_j^r \text{ and } \sigma'^-(\pi_r) = l_k^r \end{cases}$$

Thus the clockless semantics defines a possible *clockless computation* ξ^- of TCTS/CCTS as a sequence of states $\sigma_0^-, \sigma_1^-, \cdots$.

6.3 LTL formulas for Clockless Models

A remark about the LTL formulas that would be verified against clockless models, is in order. These formulas will not involve the global timing variable t. The LTL formulas will be built using finitely many

atomic propositions (constraints), which may be defined in terms of state variables for which the possible combinations of valuations needs to be finite.

Assuming that typical arithmetic constraints are defined in terms of variables in $\mathcal{U}$ (as defined before for clockless timeout and calender models), let us now define a point-wise or event based semantics for LTL formulas based on its classical semantics [CGP99]. A model for a LTL formula would consist of a sequence of states of the form

$$\sigma_0, \sigma_1, \cdots,$$

such that each state σ_i gives a boolean interpretation ($\texttt{true}, \texttt{false}$) to the propositions, and non-negative integer valued interpretation to the timeout variables in $\mathcal{T}$, timing variables in $\mathcal{X}$, and state variables in *Var*, all of which are bounded above by some positive integer constant. In a state σ_i, let us assume $\sigma_i(v)$ to be the value of $v \in \mathcal{U}$. Considering an example of an arithmetic constraint as $t_j - t_k \geq c$, where $t_j, t_k \in T \cup \mathcal{X}$ and c an integer constant, the satisfaction relation $\models$ can be defined as

$$\begin{array}{ccc}
\sigma_i \models p & \text{iff} & \sigma_i(p) = \texttt{true} \\
\sigma_i \models t_j - t_k \geq c & \text{iff} & \sigma_i(t_j) - \sigma_i(t_k) \geq c \\
\sigma_i \models \neg\phi & \text{iff} & \sigma_i \not\models \phi \\
\sigma_i \models \phi \vee \psi & \text{iff} & \sigma_i \models \phi \text{ or } \sigma_i \models \psi \\
\sigma_i \models \bigcirc\phi & \text{iff} & \sigma_{i+1} \models \phi \\
\sigma_i \models \phi \mathcal{U} \psi & \text{iff} & \exists k > i.\, \sigma_k \models \psi \text{ and } \forall j.i \leq j < k.\, \sigma_j \models \psi
\end{array}$$

In terms of these LTL formulas, using Clockless ToM, one can essentially verify all those qualitative properties of the associated real-time system, which are otherwise prohibitively difficult to do using the clocked ToM models and timed temporal logics. This is because clockless models preserve the qualitative behavior of the clocked models and LTL can effectively specify these properties. As the valuations of the variables in the clockless models are bounded, the clockless models effectively give rise to finite state behaviors. Indeed, we can also estimate the approximate size of the clockless TCTS having direct bearing on the time complexity of its LTL model-checking. Assume a clockless ToM with n parallel processes with k local timing variables. Let the valuations of timeouts and timing variables be bounded above by $M = \max(\mathcal{M})$. Also let the sizes of the clockless TTDs of these processes are bounded by D. In terms of these, the size of the clockless TTS could be bounded by $\mathcal{F} = \mathbf{O}(\max\{M^{n+k}D^n, |\Gamma^-|\})$, using asymptotic notation. This, in turn implies that complexity of model checking such clockless TTS for a LTL formula ϕ would be $\mathbf{O}(\mathcal{F}2^{|\phi|})$ [VaW86].

6.4 Clockless Models (Bi-)Simulate Clock Models

In this section we will show that clockless models (bi-)simulate clock models with respect to LTL formulas. Let us consider a ToM P and its TCTS $S_P = (\mathcal{V}, \Sigma, \Sigma_0, \Gamma)$ and also the clockless ToM P^- and corresponding timeout based clockless transition system $S_P^- = (\mathcal{V}^-, \Sigma^-, \Sigma_0^-, \Gamma^-)$; both of them modeling the same system. Given a computation $\xi : \sigma_0 \rightarrow \sigma_1 \rightarrow \cdots$ over S_P let us generate a clockless computation as a sequence of states $\sigma_0^-, \sigma_1^- \cdots$ over S_P^- as follows:

- Initial states correspond:

$$\begin{array}{l}
\forall \tau \in \mathcal{T}.\ \sigma_0^-(\tau) = \sigma_0(\tau), \\
\forall x \in \mathcal{X}.\ \sigma_0^-(x) = \sigma_0(x), \\
\sigma_0^-(\pi_i) = \sigma_0(\pi_i) = \perp .
\end{array}$$

- Entry transition: if $(\sigma_0, \sigma_1) \in \Gamma_e$ then

$$\left\{\begin{array}{ll}
1. & \forall \tau \in \mathcal{T}.\sigma_1^-(\tau) = \sigma_1(\tau), \\
2. & \forall x \in \mathcal{X}.\sigma_1^-(x) = \sigma_1(x), \\
3. & \sigma_1^-(\pi_i) = \sigma_1(\pi_i) = l_0^i
\end{array}\right.$$

- Time progress transition: if $(\sigma_{i-1}, \sigma_i) \in \Gamma_+$ then

$$\left\{\begin{array}{ll}
1. & \forall \tau \in \mathcal{T}.\sigma_i^-(\tau) = \sigma_i(\tau) - \min\{\sigma_{i-1}(\mathcal{T})\}, \\
2. & \forall x \in \mathcal{X}.\sigma_i^-(x) = \sigma_i(x), \\
3. & \forall i.\sigma_i^-(\pi_i) = \sigma_i(\pi_i).
\end{array}\right.$$

- Timeout increment transition: if $(\sigma_{i-1}, \sigma_i) \in \Gamma_e$ (which is labeled by the instruction $\rho \Rightarrow \langle \tau_i := update_i^-, \gamma, f \rangle$) then

$$\begin{cases} 1. & \text{if } \sigma_{i-1}^-(\tau_i) = 0 \text{ then } \sigma_i^-(\tau_i) = update_i^-, \text{ else } \sigma_i^-(\tau_j) = \sigma_i(\tau_j) \\ 2. & \forall x \in \gamma. \sigma_i^-(x) = \min\{\sigma_{i-2}(\mathcal{T})\} + \sigma_{i-1}(x), \\ 3. & \forall x \in \mathcal{X} \setminus \gamma. \sigma_i^-(x) = \sigma_{i-1}(x), \\ 4. & \forall i. \sigma_i^-(\pi_i) = \sigma_i(\pi_i). \end{cases}$$

  where $update_i^-$ is defined in P^-.

- Synchronous communication: if $(\sigma_{i-1}, \sigma_i) \in \Gamma_{syn\_comm}$ then

$$\begin{cases} 1. & \sigma_i^-(\tau_s) = \sigma_i(\tau_s) \text{ and } \sigma_i^-(\tau_r) = \sigma_i(\tau_r) \\ 2. & \forall x \in \mathcal{X}. \sigma_i^-(x) = \sigma_i(x) \\ 3. & \sigma_i^-(\bar{m}) = \sigma_i(\bar{m}) \text{ and } \sigma_{i-1}^-(m) = \sigma_{i-1}(m) \\ 4. & \forall v \in G \cup L_s : \sigma_i^-(v) = \sigma_i(v) \text{ and } \forall v \in G \cup L_r : \sigma_i^-(v) = \sigma_i(v) \\ & \forall v \in Var \setminus (G \cup L_s \cup L_r) : \ \sigma_i^-(v) = \sigma_i(v) \\ 5. & \sigma_{i-1}^-(\pi_s) = \ \sigma_{i-1}(\pi_s) = \ l_j^s, \sigma_{i-1}^-(\pi_r) = \ \sigma_{i-1}(\pi_r) = \ l_j^r \text{ and} \\ & \sigma_i^-(\pi_s) = \ \sigma_i^-(\pi_s) = \ l_k^s, \sigma_i^-(\pi_r) = \ \sigma_i(\pi_r) = \ l_k^r \end{cases}$$

Check that $\sigma_0^- \in \Sigma_0^-$ and $\forall i. (\sigma_{i-1}^-, \sigma_i^-) \in \Gamma^-$. It is clear $\xi^- = \sigma_0^- \rightarrow \sigma_1^- \rightarrow \ldots$ forms a clockless computation over S_P^-. We can associate a mapping $\mathrm{Tr} : \Sigma \times \Sigma \rightarrow \Sigma^-$ parameterized by an entry transition as follows. Fix two states, $\sigma_0 \in \Sigma_0, \sigma_1 \in \Sigma$, such that $(\sigma_0, \sigma_1) \in \Gamma_e$. Call $\gamma = (\sigma_0, \sigma_1)$. Then define $\mathrm{Tr}_\gamma(\sigma_0, \sigma_0) = \sigma_0^-, \mathrm{Tr}_\gamma(\sigma_i, \sigma_{i-1}) = \sigma_i^-, \forall i \geq 1$.

We say that computations $\xi : \sigma_0 \sigma_1 \ldots$ in S_P and $\xi^- : \sigma_0^- \sigma_1^- \ldots$ in S_P^- correspond if and only if there exists $\mathrm{Tr}_\gamma : \Sigma \times \Sigma \rightarrow \Sigma^-$ such that $\sigma_0^- = \mathrm{Tr}_\gamma(\sigma_0, \sigma_0)$ and for every $i \geq 0$, $\sigma_i^- = \mathrm{Tr}_\gamma(\sigma_i, \sigma_{i-1})$, where $\gamma = (\sigma_0, \sigma_1)$. Let $\sigma \in \Sigma$ and $\sigma^- \in \Sigma^-$ be two states and there be a computation in S_P which starts in σ. Then it is easy to see that there exists a corresponding computation in S_P^- beginning with σ^- [CGP99].

We consider LTL formulas consisting of propositions and variables appearing in clockless transition system of S_P^-. Assume $\sigma \in \Sigma$ and $\sigma^- \in \Sigma^-$ are two states such that $\mathrm{Tr}_\gamma(\sigma, \sigma') = \sigma^-$ for some $\sigma' \in \Sigma$ and some entry transition γ. Then for any LTL formula ϕ, $\sigma^- \models \phi$ implies $\sigma \models \phi$ (using the semantics of LTL formulas as discussed in Section 6.3). This can be proved using the induction on the structure of ϕ. Finally, $S_P^- \models \phi$ implies $S_P \models \phi$. This is in some sense, we can say S_P^- simulates S_P [CGP99]. Thus it is enough to verify properties on the clockless transition system S_P^- instead of on S_P.

Similar results can be established for calendar-based clocked transition system (CCTS) also. In fact a reverse mapping cane be defined too. To see this let us assume $\xi^- = \sigma_0^-, \sigma_1^- \ldots$ to be a clockless computation over S^-. Now generate a sequence of states $\sigma_0, \sigma_1 \ldots$ as follows.

- $\sigma_0(t) = \min\{\sigma_0^-(\mathcal{T})\}, \forall \tau \in \mathcal{T}. \sigma_0(\tau) = \sigma_0^-(\tau), \forall x \in \mathcal{X}. \sigma_0(x) = \sigma_0^-(x), \sigma_0(\pi_i) = \sigma_0^-(\pi_i) = \perp$.

- $$\text{if } (\sigma_0^-, \sigma_1^-) \in \Gamma_e \text{ then } \begin{cases} 1. & \forall \tau \in \mathcal{T}. \sigma_1(\tau) = \sigma_1^-(\tau), \\ 2. & \forall x \in \mathcal{X}. \sigma_1(x) = \sigma_1^-(x), \\ 3. & \sigma_1(\pi_i) = \sigma_1^-(\pi_i) = l_0^i, \\ 4. & \sigma_1(t) = \sigma_1^-(t) = \sigma_0^-(t) \end{cases}$$

- $$\text{if } (\sigma_{i-1}^-, \sigma_i^-) \in \Gamma_+ \text{ then } \begin{cases} 1. & \forall \tau \in \mathcal{T}. \sigma_i(\tau) = \sigma_{i-1}(\tau), \\ 2. & \forall x \in \mathcal{X}. \sigma_i(x) = \sigma_{i-1}(x), \\ 3. & \forall i. \sigma_i(\pi_i) = \sigma_i^-(\pi_i), \\ 4. & \sigma_i(t) = \min\{\sigma_{i-1}(\mathcal{T})\}. \end{cases}$$

- $$\text{if } (\sigma_{i-1}^-, \sigma_i^-) \in \Gamma_e \text{ then } \begin{cases} 1. & \text{if } \sigma_i^-(\tau_i) = 0 \text{ then } \sigma_i(\tau_i) = update_i, \text{ else } \sigma_i(\tau_j) = \sigma_{i-1}(\tau_j) \\ 2. & \forall x \in \mathcal{X}. \sigma_i(x) = \sigma_{i-1}(x), \\ 3. & \forall i. \sigma_i(\pi_i) = \sigma_i^-(\pi_i), \\ 4. & \sigma_i(t) = \sigma_{i-1}(t) \end{cases}$$

•

$$\text{if } (\sigma^-_{i-1}, \sigma^-_i) \in \Gamma_{syn\_comm} \text{ then } \begin{cases} 1. & \sigma_i(\tau_s) = \sigma^-_i(\tau_s) \text{ and } \sigma_i(\tau_r) = \sigma^-_i(\tau_r) \\ 2. & \forall x \in \mathcal{X}.\sigma_i(x) = \sigma^-_i(x) \\ 3. & \sigma_i(\bar{m}) = \sigma^-_i(\bar{m}) \text{ and } \sigma_{i-1}(m) = \sigma^-_{i-1}(m) \\ 4. & \forall v \in G \cup L_s : \sigma_i(v) = \sigma^-_i(v) \text{ and } \forall v \in G \cup L_r : \sigma_i(v) = \sigma^-_i(v) \\ & \forall v \in Var \setminus (G \cup L_s \cup L_r) : \ \sigma_i(v) = \sigma^-_i(v) \\ 5. & \sigma_{i-1}(\pi_s) = \ \sigma^-_{i-1}(\pi_s) = \ l^s_j, \sigma_{i-1}(\pi_r) = \ \sigma^-_{i-1}(\pi_r) = \ l^r_j \text{ and} \\ & \sigma^-_i(\pi_s) = \ \sigma^-_i(\pi_s) = \ l^s_k, \sigma^-_i(\pi_r) = \ \sigma_i(\pi_r) = \ l^r_k, \\ 6. & \sigma_i(t) = \sigma_{i-1}(t). \end{cases}$$

Clearly, $\xi : \sigma_0 \rightarrow \sigma_1 \rightarrow \cdots$ is a computation over S. Associate a mapping $\mathrm{Tr}' : \Sigma^- \rightarrow \Sigma$ with this such that $\mathrm{Tr}' : \sigma^-_i \mapsto \sigma_i, \forall i$. Let us try to compose these two mappings. Note that $\mathrm{Tr} \circ \mathrm{Tr}' = \mathrm{id}, \mathrm{Tr}' \circ \mathrm{Tr} = \mathrm{id}$ where id is an identity mapping. This implies that Tr is a bijective mapping and $(\mathrm{Tr})^{-1} = \mathrm{Tr}'$.

Define a relation $\mathcal{B} \subseteq \Sigma \times \Sigma^-$ as follows: for two states $s \in \Sigma$ and $s^- \in \Sigma^-$ we have $\mathcal{B}(s, s^-)$ if and only if $s^- = \mathrm{Tr}(s)$. Assume s and s^- satisfy the same atomic propositions. Also observe that

- for every state $s_1 \in \Sigma : (s, s_1) \in \Gamma$ there exists $s^-_1 \in \Sigma^- : (s^-, s^-_1) \in \Gamma^-$ such that $s^-_1 = \mathrm{Tr}(s_1)$, *i.e.,* $\mathcal{B}(s^-, s^-_1)$.
- for every state $s^-_1 \in \Sigma^- : (s^-, s^-_1) \in \Gamma^-$ there exists $s_1 \in \Sigma : (s, s_1) \in \Gamma$ such that $s_1 = (\mathrm{Tr})^{-1}(s^-_1)$, *i.e.,* $\mathcal{B}(s^-, s^-_1)$.

Hence $\mathcal{B}$ is a bisimulation relation between S and S^-. Finally, we can see for this bisimulation relation $\mathcal{B}$, for every initial state $s_0 \in \Sigma$ in S there is an initial state $s^-_0 \in \Sigma^-$ in S^- such that $cB(s_0, s^-_0)$. In addition, for every initial state $s^-_0 \in \Sigma^-$ in S^- there is an initial state $s_0 \in \Sigma$ in S such that $\mathcal{B}(s_0, s^-_0)$. Hence S and S^- are bisimulation equivalent [CGP99]. Since bisimulation equivalent structures preserve LTL formulas [CGP99] we shall be dealing with clockless timeout based models for our verification purposes.

7 Experimental Evaluation

In this section we illustrate finite state verification of real-time systems through clockless modeling on three real-time protocols introduced earlier - Fisher's Mutual Exclusion Protocol, TGC, and TTA startup protocol. We perform finite state model checking of these protocols by Spin and SAL-smc model checkers. For applying our technique we assume that the timeout increments of these protocols are more than one time unit. We carry out our experiments on a machine with 2.26GHz Intel Core 2 Duo processor, 3 MB shared level 2 cache and 2GB 1066MHz DDR3 SDRAM, running MAC OS X Version 10.5.7. For experimentation with Spin, we use XSpin graphical interface. To verify a property prop for a SAL specification *model.sal* we use the following SAL command:

sal-smc -v 3 model prop –enable-dynamic-reorder

Here enable-dynamic-reorder is a flag used with SAL-smc that enables dynamic reordering of BDD variables.

7.1 Fischer's Mutual Exclusion Protocol

A clockless model of the Fisher's mutual exclusion protocol is depicted in Figure 4. We consider the following `safety` property for Fischer's protocol, *"no more than one processor can be in the critical region at any time"*. The property is frequently referred as `mutual exclusion` property. This can be represented in LTL as:

$$\Box(in\_critical \ \leq 1)$$

To verify the `safety` property for Fischer's mutual exclusion protocol in Spin we used *exhaustive verification* and *bitstate hashing* technique available in Spin, in both the cases keeping the the option of *partial order reduction* turned on. By *exhaustive verification* technique, we could verify models containing only upto 4 nodes. *Bitstate hashing* enabled us to verify the same property for models with upto 6 nodes. Table 1 illustrates the computational resources and time required to prove the `safety` property for Fischer's mutual exclusion protocol using *bitstate hashing* technique.

We perform clockless modeling of Fischer's protocol in SAL language. Table 2 presents the number of states visited and time required to prove the `mutual exclusion` property. We have been able to verify

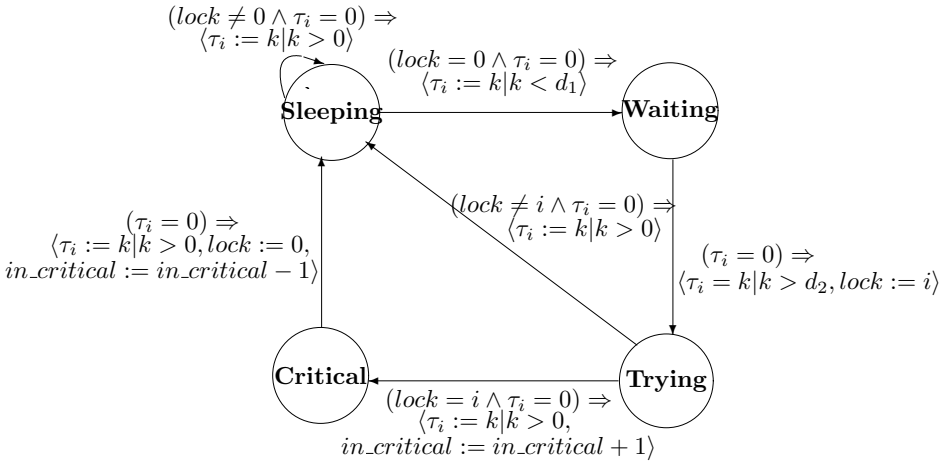

Figure 4: Clockless model for the i^{th} processor in the Fischer's Protocol

Table 1: Computational resources required for verification of the Fischer's Protocol using *bitstate hashing*

Property	N	# States Stored	# States Matched	# Transitions	Memory (MB)	Time (sec)
Safety	2	563	463	1026	8.501	0.1
	3	18220	29625	47845	8.598	0.11
	4	667995	1716011	2384003	44.383	5.01
	5	21373206	75073507	96446713	395.366	203.09
	6	36720364	1.4129329e+08	1.7801365e+08	1722.014	908.56

Table 2: States explored and time required to verify mutual-exclusion property by SAL-smc for Fischer's protocol

# Nodes	# States Explored	Time (sec)	# Nodes	# States Explored	Time (sec)
2	468	0.15	10	1.189e12	69.9
3	7968	0.30	11	1.697e13	213.76
4	124760	1.40	12	2.417e14	196.36
5	1.876e6	2.35	13	3.438e15	2767.91
6	2.760e7	3.84	14	4.885e16	21731.91
7	4.010e8	12.43	15	6.935e17	4516.85
8	5.786e9	23.04	16	9.839e18	10376.53
9	8.306e10	44.604	17	–	–

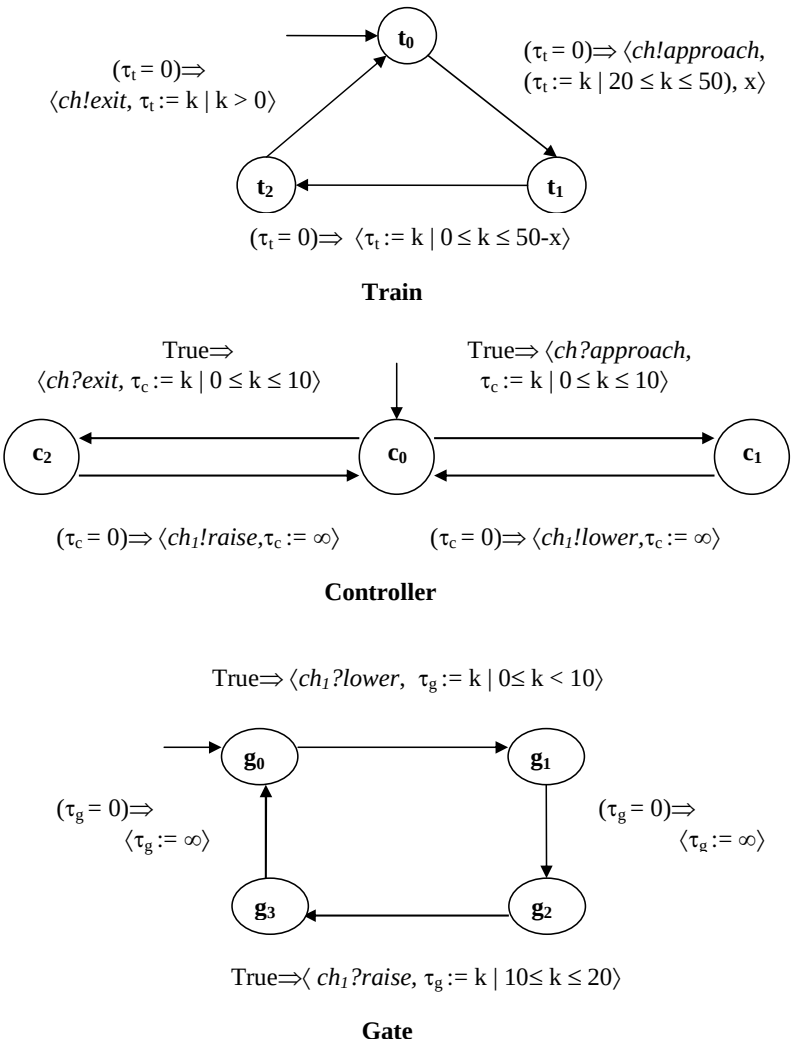

Figure 5: Clockless model for Train-Gate Controller

the `mutual exclusion` property for Fischer's protocol with 16 processors in around 3 hours (except the model for 14 nodes, which took around 6 hours). We tried to verify the protocol for 17 and 18 nodes, and in both the cases, verification ran for more than 7 hours. We did not go for higher number of nodes.

The Fisher's protocol has been verified under dense time for the same `mutual exclusion` property in [DuS04a]. A direct attempt to prove the property by k-induction with induction depth up to 15 fails for even 2 processors. However, using a sequence of lemmas it was possible to prove the property by induction at depth 1 for upto 13 processors for the same SAL specification (Table 3.1 of [DuS04a]). The property was also proved by induction by a sequence of lemmas for a *different* SAL specification for a maximum number of 53 processors (Table 3.5 of [DuS04a]).

To compare the performance and scalability of our verification approach with UPPAAL, we verified Fischer's mutual exclusion protocol available with UPPAAL distribution. The UPPAAL model is based on the framework of *timed automaton*. The `mutual exclusion` property could be verified successfully for up to 12 nodes. For 13 nodes, the verification process did not stop even in 7 hours. In verification with UPPAAL, the TA is reduced to the zone automata which are finite representations of infinite state systems. Although both our clockless verification scheme and UPPAAL's zone automata based verification are based on abstracting an infinite system to a finite one, this experimental result shows that our technique is more scalable than UPPAAL, while using SAL-smc model checker.

Table 3: Computational resources and time required for verification of the Train-Gate Controller under exhaustive verification

Properties	# States Stored	# States Matched	# Transitions	Memory (MB)	Time (sec)
`Safety`	246236	422596	668832	47.947	1.50
`Timeliness`	253500	415484	668984	50.389	1.58

Table 4: States explored and time required to verify `safety` and `timeliness` properties by SAL-smc for TGC

Properties	# States Explored	Time (sec)
`Safety`	1.123e6	5.24
`Timeliness`	4.807e5	2.41

7.2 Train-Gate Controller

A clockless model of TGC is depicted in Figures 5.For the TGC example as discussed before, we consider `safety` and `timeliness` properties for verification. The `safety` property says: *When the Train crosses the line, the Gate should be down.* The property is expressed in LTL as:

$$\Box((t\_state = t_2) \Rightarrow (g\_state = g_2))$$

where, t_state denotes different states of the Train, and it is t_2, when it comes into the crossing, g_state denotes different states of Gate, and is g_2, when the Gate is down.

`Timeliness` property, in general ensures that the time between two states will by bounded by a particular value. We can find many `timeliness` properties in this example. We select an important one, *"the time between the transmission of the approach signal by the Train and when the Gate is down should not be more than* 20 *time units"*. To verify this property we use two auxiliary flags, $flag_1$ and $flag_2$ in our model. When the first event occurs $flag_1$ is set as $true$. When the second event happens, $flag_2$ is set as $true$ and $flag_1$ is reset to $false$.

A global variable $time\_diff$ initially set to 0, captures the time between the instants when two flags are set. During every discrete transition between the two discrete transitions of interest, minimum timeout value is added to $time\_diff$. The `timeliness` property is then specified as follows, *"the value of* $time\_diff$ *never goes above 20"*. This is expressed in LTL as,

$$\Box(time\_diff \ \leq \ 20)$$

In Table 3, we illustrates computational resources and time required to prove the `safety` and the `timeliness` properties for TGC by Spin model checker. Both the properties have been proved by *exhaustive verification* keeping the the option of *partial order reduction* turned on.

We verify the `safety` and `timeliness` properties for TGC by SAL-smc, and the result is shown in Table 4.

It may be noted that dense time verification of the `safety` property for TGC took 46.15 seconds [DuS04a]. This was proved by k-induction at depth 14 using SAL-inf-bmc.

7.3 TTA Startup Algorithm

Figure 6 depicts the clockless model for the TTA startup algorithm as discussed before in the Section 4.2. We consider the following `safety` property, *"whenever any two nodes are in their* active *state the nodes agree on the slot time"*. For two nodes participating in the startup process, the corresponding LTL property is given below:

$$\Box((p_1 \wedge p_2) \wedge (q_1 \wedge q_2) \ \Rightarrow \ \Diamond(r \wedge s)),$$

where $p_1 \equiv (pc[0] = state\_active), p_2 \equiv (pc[1] = state\_active), q_1 \equiv (time\_out[0] > 0), q_2 \equiv (time\_out[1] > 0), r \equiv (time\_out[0] = time\_out[1]), s \equiv (slot[0] = slot[1])$. Also, $pc[i]$ denotes the current state of the

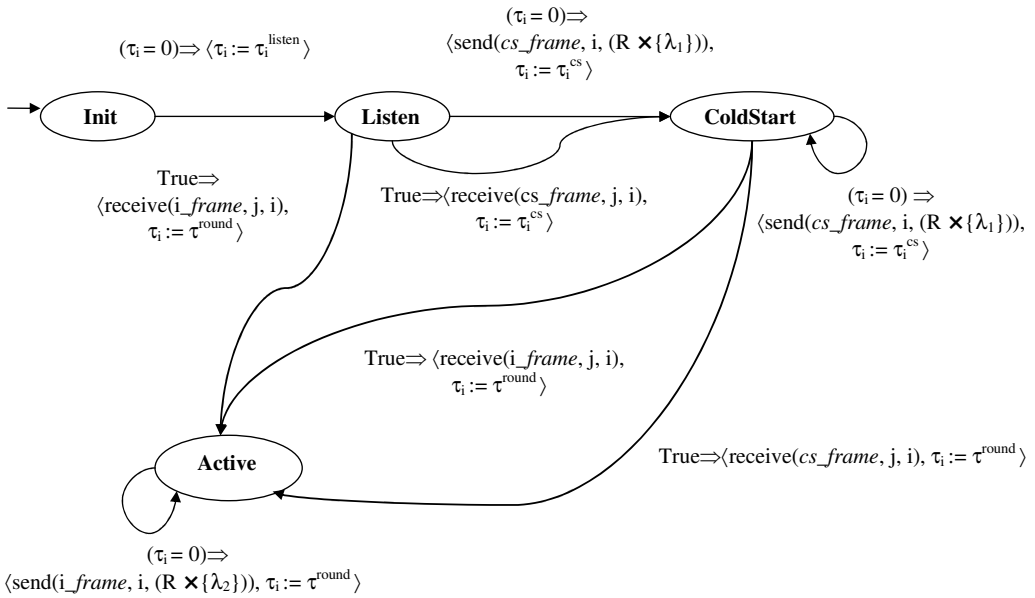

Figure 6: Clockless model for the i^{th} processor in TTA Startup algorithm.

i^{th} node, $time\_out[i]$ denotes the timeout of the i^{th} node, and $slot[i]$ denotes the current time slot viewed by the i^{th} node. *state_active* characterizes the *synchronized* state of a node.

The `safety` property ensures that when the nodes are in *active* state, then they are indeed synchronized. But it does not address the question whether all the nodes will be eventually synchronized or not. To ensure that this happens, it is specified in the form of the following `liveness` property, *"eventually all the nodes will be in active state and continue to do so"*. This `liveness` property for two nodes can be specified in LTL as follows:

$$\Diamond\Box((pc[0] = state\_active) \wedge (pc[1] = state\_active))$$

To verify the `safety` and the `liveness` property for TTA startup in Spin, we use both *exhaustive verification* and *bitstate hashing* techniques with *partial order reduction* availed. By *exhaustive verification* technique, the `safety` property can be verified for TTA models containing upto 5 nodes, and the `liveness` property can be verified upto 4 nodes. *Bitstate hashing* enables us to verify both the properties for models with upto 9 nodes. For 10 nodes, the verification does not terminate even in 4 hours. Table 5 illustrates the computational resources and time required to prove the `safety` and `liveness` properties for TTA Startup protocol using *bitstate hashing* technique.

In Table 6 we describe the number of states and time required to prove the `safety` and `liveness` properties for the TTA Startup protocol using SAL-smc. We have been able to verify both `safety` and `liveness` properties for TTA startup protocol for upto 8 nodes in around 1 hour. Let us contrast our verification effort with the dense time modeling and verification of the same protocol reported in [DuS04a, DuS04b]. Using bounded model checking the same `safety` property was proved for only 2 nodes by k-induction at depth 8, that too using 3 auxiliary lemmas (the proof failed for 3 nodes). However, the invariant can be strengthened by constructing an abstraction of the transition systems using a verification diagram-based approach [Rus00], and subsequently the property was verified for upto 10 nodes.

8 Extension of Timeout and Calendar based Models

In this section we extend our model to incorporate other modeling concepts like inter-process scheduling, priorities and interrupts, and urgent and committed locations. These extensions will be illustrated using ToM as a base model, however they can be easily adapted for calendar based ToM also. Also note that the digitization result presented in Section 5.3, and the finitary reduction and associated clockless modeling proposed in Section 6 are applicable to these extended models as well because the additional components defined in these (extended) models are independent of the variables present in the base model and therefore, do not affect the underlying semantics of the base model.

Table 5: Computational resources and time required to verify `safety` and `liveness` property by bitstate hashing technique in Spin for TTA Startup

Properties	N	# States Stored	# States Matched	# Transitions	Memory (MB)	Time (sec)
`Safety`	2	487	143	630	8.501	0.01
	3	6142	6490	12632	8.501	0.05
	4	217852	483497	701349	8.501	1.46
	5	4126813	13188075	17314888	8.501	34.72
	6	16508262	62593403	79101665	8.501	165.46
	7	34442659	1.2702415e+08	1.6146681e+08	8.501	364.99
	8	40175448	2.4473144e+08	2.8490689e+08	8.598	665.63
	9	41008029	1.2976237e+09	1.3386317e+09	8.598	4390.17
`Liveness`	2	725	1036	2481	8.501	0.04
	3	8305	21980	38562	8.501	0.12
	4	249439	1149753	1648373	8.501	3.75
	5	4339737	28293352	36972211	8.501	83.32
	6	12678951	1.1096373e+08	1.3851011e+08	8.501	314.08
	7	20128894	2.0273546e+08	2.4108713e+08	8.501	531.80
	8	25361336	3.4848047e+08	3.8927174e+08	8.598	936.05
	9	40305514	2.307274e+09	2.3482827e+09	8.598	7039.02

Table 6: Computational resources required to verify `safety` and `liveness` property by SAL-smc for the TTA Startup

# Nodes	# States	Time (`Safety` Property) (sec)	Time (`Liveness` Property) (sec)
2	68	0.34	1.18
3	485	0.63	3.28
4	5297	2.75	10.56
5	76345	13.11	48.31
6	1331650	77.23	563.82
7	26872795	4044.31	742.90
8	615902175	3440.63	3101.26

8.1 Modeling Inter-Process Scheduling

So far, we have considered models capturing true parallelism with non-determinism. However, in some cases the ability of a system to meet real-time constraints crucially depends on the number of processors that are available and also, on the process scheduling algorithm. Thus, we need to distinguish between the models of multiprocessing and multiprogramming. We show how ToM can be extended to include fixed number of programs that are executed by time sharing, on a single processor. Subsequently we use our framework to model priorities and interrupts for a general distributed multiprogramming system. These are motivated by the framework of multiprogramming system introduced in [HMP92b].

A Multiprogramming Timeout based Model (MToM) P has the form

$$\{\theta\}[(P_{11}|||\ldots|||P_{1l_1})||(P_{21}|||\ldots|||P_{2l_2})||\ldots||(P_{m1}|||\ldots|||P_{ml_m})],$$

where each process $P_{i1}\ldots P_{il_i}, 1 \leq i \leq m$ is a sequential non-deterministic process as we have seen before. By $P_\alpha|||P_\beta$ we mean processes P_α and P_β share a single processor and are executed on one transition at a time according to some scheduling policy. Thus there are m groups of processes in the above MToM such that all the processes in a group share the same processor, e.g., the processes $P_{11}\ldots P_{1l_1}$ would execute on the first processor. Processes in different groups running on different processors execute concurrently as in the case of ToM defined in Section 3.1.1. A special case of synchronous communication needs special care because both the processes need to be simultaneously active: If process P_{ij} and $P_{i'j'}$ have a synchronous communication, these processes must be executing on different processors, that is, $i \neq i'$.

For example, $[(P_{11}|||P_{12}|||P_{13})||(P_{21}|||P_{22})]$ is the model of a system with five processes running on two processors. The first three processes share the first processor and next two the second processor. A synchronous communication can take place between two processes only when these processes belong to different groups.

A timed transition system $S_P = (\mathcal{V}, \Sigma, \Sigma_0, \Gamma)$ can be associated with an MToM also. The key difference now is that $\mathcal{V}$ contains additional processor control variables $\mu_1, \ldots, \mu_m$, such that μ_i ranges over $\{1, \ldots, l_i, \perp\}$, *i.e.*, $\mathcal{V} = \mathcal{U} \cup \{\mu_1, \pi_{11}, \ldots, \pi_{1l_1}\} \cup \{\mu_2, \pi_{21}, \ldots, \pi_{2l_2}\} \cup \ldots \cup \{\mu_m, \pi_{m1}, \ldots, \pi_{ml_m}\}$. The processor control variables assume the value $\perp$ before the processor starts executing the processes in a group. Thereafter, the control of the process $P_{i\mu_i}$ resides at the location π_{μ_i} executing on the i^{th} processor. In other terms, only the process $P_{i\mu_i}$ is active on the i^{th} processor, while all other processes $P_{ij}, j \neq \mu_i$ are suspended. When the execution of the process $P_{i\mu_i}$ is suspended as per the scheduling policy, in future it can only resume at the last suspended location $\pi_{i\mu_i}$.

For simplicity, we will next consider the case of a single processor, that is $m = 1$ and will drop the subscript 1 in the notations e.g., μ would stand for μ_1 and π_j for π_{1j}. Let us now discuss some of the transitions that would additionally occur in this framework. For example, Γ will contain a set of scheduling transitions, Γ_{sch}.

A scheduling policy determines the set of scheduling transitions. We consider only scheduling policies with a single entry transition, that is enabled on all states. The entry transition is assumed to be enabled on the initial states, and activates non-deterministically one of the competing processes. A very popular and simple scheduling policy is based on *greedy scheduling*. According to which, a process, currently in the control of the processor, continues to remain active until all its transition are disabled, when an arbitrary (other) process with an enabled transition takes over. More flexible scheduling strategies can be implemented by incorporating explicit scheduling instruction $resume(s)$, where $s \subset \{1, \ldots, n\}$ determines a subset of processes. The scheduling operation $resume(s)$ suspends the currently active process, P_i and activates, nondeterministically, one of the processes P_j, with $j \in s$. A scheduling edge in the process P_i will be represented as:

$$l^i_j \xrightarrow{\rho \Rightarrow \langle resume(s), [l,m] \rangle} l^i_k$$

Where $[l, m], l < m$ specifies (optional) delay which the scheduling operation may take between l and m time units. Such an edge introduces an additional transition in Γ, and grouped in Γ_{sch} as follows:

$$\nu_{sch} \equiv (\sigma, \sigma') \in \Gamma_{sch} \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) + \delta \\ 3. & \forall y \in \mathcal{V} \setminus \{\mu, \pi_i\} : \sigma'(y) = \sigma(y) \\ 4. & \sigma(\mu) = i \text{ and } \sigma'(\mu) \in s \\ 5. & \sigma(\pi_i) = l^i_j \text{ and } \sigma'(\pi_i) = l^i_k \end{cases}$$

Where δ is a randomly selected constant such that $l \leq \delta \leq m$. To add, a $suspend(i, j)$ operation, which suspends a process P_i and activates process P_j, can also be defined as $resume(\{1 \leq j \leq m \mid i \neq j\})$, that

is, the instruction $suspend(i, j)$ delegates the control from the currently active process P_i to the process P_j. In practice, processes P_i and P_j could have some operational relationship with each other, e.g., P_i is the parent process, which spawns P_j as its its child process, goes into waiting state and activates P_j. On termination P_j may hand over the control back to P_i using the operation $resume(\{i\})$.

8.2 Modeling Priorities and Interrupts

We will next discuss how interrupts can be handled by way of introducing static priorities with global preemption semantics. Priorities will be represented using non negative integers and will be assigned to every transition such that lower value would be interpreted as higher priority. During execution a transition with the highest priority at any time point is selected and current process would be suspended if the ready process having the transition with the highest priority happens not to be the current process. A Multiprogramming Timeout based Model (MToM) P with priority is one in which a priority is associated with every transition in the timed transition systems for P. Using priorities it is possible to design a simple, static scheduling strategy without resorting to explicitly constructing a scheduler.

As an example, in a ToM, an extended timeout edge $e : (l_j^i, \rho_e \ \Rightarrow \langle \tau_i := update_i, \gamma, f, \mathbf{p_e} \rangle, l_k^i)$ in the graph of the process P_i would be represented as

$$e : l_j^i \ \overset{\rho_e \ \Rightarrow \langle \tau_i := update_i, \gamma, f, \mathbf{p_e} \rangle}{\longrightarrow} \ l_k^i,$$

where an additional parameter $\mathbf{p_e} \in \mathbb{N}$ is the priority associated with the transition e. All other edges e.g., synchronous communication and asynchronous communication would be extended similarly.

Accordingly, we extend the semantics also. For the prioritized timeout edges, a transition with the highest priority is allowed by adding it in Γ_0 in the following way.

Prioritized Timeout Increment Transition: Collect all those extended timeout edges e for which corresponding transitions are enabled in the current state σ, that is, ρ_e *holds in* σ. Let En_σ be the set of these enabled edges. Now select those timeout edges $e_h \in En_\sigma$, which have the highest priority, i.e., $\forall e' \in En_\sigma . \mathbf{p_h} \leq \mathbf{p_{e'}}$. Add transition $\nu_h \equiv (\sigma, \sigma')$ in Γ_0 such that:

$$\nu_h \equiv (\sigma, \sigma') \in \Gamma_0 \Leftrightarrow \begin{cases} 1. & \rho_h \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \textbf{If } \sigma(\tau_i) = \sigma(t) \\ & \quad \textbf{then } \sigma'(\tau_i) = update_i > \sigma(\tau_i) \\ & \textbf{else } \sigma'(\tau_i) = \sigma(\tau_i) \\ 4. & \forall y \in \gamma : \ \sigma'(y) = \sigma_e(t) \ \text{ and} \\ & \forall x \in \mathcal{X} \setminus \gamma : \ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in G \cup L_i : \sigma'(v) = f(\sigma(v)) \text{ and} \\ & \forall v \in Var \setminus (G \cup L_i) : \ \sigma'(v) = \sigma(v) \\ 6. & \sigma(\pi_i) = \ l_j^i \text{ and } \sigma'(\pi_i) = l_k^i \\ 7. & \sigma'(\mu) = i \end{cases}$$

If there are multiple enabled edges with the same highest priority, their corresponding transitions are non deterministically interleaved.

The remaining all other transitions can also be extended similarly. Under such extended syntax and semantics, an interrupt can be modeled as an edge having relatively high priority than other enabled transitions:

$$e_{int} : (l_j^i, True \ \Rightarrow \langle \tau_i := update_i, f, \mathbf{p_{int}} \rangle, l_k^i)$$

where $update_i$ specifies the delay in interrupt processing and f specifies the steps in interrupt processing. Note $\mathbf{p_{int}}$ is such that $\forall \sigma \in \Sigma . \forall e \in En_\sigma . \mathbf{p_{int}} \leq \mathbf{p_e}$.

8.3 Modeling Urgent Location and Committed Location

In UPPAAL there are three different types of locations: *normal locations*, *urgent locations* and *committed locations* [BDL04]. In a normal location time can progress, but in urgent and committed locations time is not allowed to proceed. Moreover, there is a subtle difference between urgent and committed locations. Urgent locations can be interleaved with the normal locations, but a committed location has to be followed by its immediate successor. The requirement of considering a location to be urgent or committed arises out of the nature of the application being modeled in UPPAAL. For example, committed locations are

used to model atomic behaviors in multi-way synchronizations and atomic broadcasting in real-time systems [BGK02].

In timeout and calendar based models, we model an urgent or a committed location in the following way. For all the incoming edges to the the urgent or committed location in process P_i, $update_i$ is set to current time t, and in case of clockless modeling $update_i$ is set to 0.

If a process in a system has a committed location, we introduce a boolean variable $committed\_flag$ in the set of global variables G. For all the incoming edges to a committed location, $committed\_flag$ is set to 1 (part of f) and for incoming edges to a non-committed state one is not allowed to set the flag to 1. The guard ρ for a transition following a committed location is always $True$ and $committed\_flag$ is reset during this transition. For all the transitions except those following the committed locations, the existing guard ρ is replaced by $\rho \wedge (committed\_flag \neq 1)$. This will not allow any other process to take a discrete transition when a process is in a committed state.

9 Conclusion and Further Work

In this work we have considered the well-known problem of real-time verification with dense time dynamics using timeout and calendar based models and proposed a technique to simplify this to a finite state verification problem. Towards this, we define a specification formalism for these models as timeout transition diagrams with associated transition system semantics. Next, we proposed a two-step reduction technique for rendering these models amenable to finite state verification under discrete dynamics. Our experimental results bring out the advantages gained by this technique over infinite state modeling and verification. Experiments on Fisher's protocol and TTA startup protocol highlight that the verification technique scales reasonably well. Further, liveness properties can be verified in this framework, which is beyond the capability of infinite state verification. Though in [DuS04a], it has been reported that verification of Fischer's protocol can be scaled up to 53 nodes, the verification process involved finding out auxiliary lemmas manually, which is a non-trivial process. On the other hand our finite state verification, though could not be scaled to this extent, is nonetheless simple and straight-forward. The verification effort involves only modeling the protocols faithfully. SAL offers a number of tools for finite state verification, for example, SAL-sim, SAL-path-finder and SAL-deadlock-checker, which help quite a lot in the verification process. Such tool support is yet not available for infinite state verification. Moreover, one can use any finite state verification engine of choice using our framework.

We limited our attention to the qualitative temporal properties that exclusively corresponds to LTL formulas. However, the proposed reduction technique is amenable to any specification logic which is closed under inverse digitization including branching time temporal logics CTL or CTL*.

The effectiveness of the proposed finitary reduction technique can be further scaled up by integrating it with additional abstraction techniques to verify parametric systems, with arbitrary but finite number of identical processes. Saïdi and Lesens [LeS97] presented an algorithm for automatically constructing abstraction for such systems to verify `safety` properties. The $(0, 1, \infty)$ counter abstraction method proposed in [PXZ02] deals with the verification of `liveness` properties by abstracting a parameterized system of unbounded size into a finite-state system. The proposed formalism can be further optimized by considering timeouts as shared variables among processes, so that timeout updation rules could specify new timeout values based upon those of other processes in the system. This optimization would increase the level of synchronization between component processes and would hopefully scale up the models.

In the larger perspective it can be said that for most of the timeout and calendar based models (i.e., for which timeout updates are not restricted to $(0, 1)$-interval) verification of LTL properties with dense time dynamics reduces to finite state modeling and verification of the same properties. In industrial designs, this could offer a significant advantage as it is easier for practitioners to use finite state model checkers to model and verify timed systems.

Decidability and complexity theoretic aspects of the reachability analysis on these models is an important research direction for further investigation. A comparison of expressiveness of ToM (or calender based ToM) with other known formal models of real-time systems including Timed Automata [Alu99], Timed Petri Nets [Jia98], and Timed Process Algebras [BeJ91] would shed light on the comparative strength of these models for practical purposes. For example, these comparisons could reveal other properties desirable of a modeling framework including compositionality, robustness against clock drifts, and may demonstrate the difficulty of modeling timeout models using these models as compared to ToM.

Acknowledgment Indranil Saha and Suman Roy did most of this work when they were with HTS Research, Bangalore.

References

[Alu99] R. Alur. "Timed Automata". In *Proceedings of International Conference on Computer-Aided Verification (CAV'99)*, LNCS 1633, Springer-Verlag, pp. 8-22, 1999.

[AlD94] R. Alur and D. L. Dill. "A Theory of Timed Automata." In *Theoretical Computer Science*, vol. 126, number 2, pp. 183-235, 1994.

[AlH91] R. Alur and T.A. Henzinger. "Logics and Models of Real Time: A Survey." In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pp. 74-106, Springer, 1991.

[BMN00] P. Bellini, R. Mattolini, and P. Nesi. "Temporal Logics for Real-Time System Specification." In *ACM Computing Surveys*, vol. 32, number 1, 2000.

[BeJ91] Jos C. M. Baeten and Jan A. Bergstra. "Real Time Process Algebra." In *Formal Aspects of Computing*, vol. 3, number 2, pp. 142–188, 1991.

[BDL04] Behrmann G., David A. and Larsen K. G.. "A Tutorial on UPPAAL". In *4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04)*, LNCS 3185, Springer-Verlag, pp. 200–236, 2004.

[BGK02] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi. "Automated Analysis of an Audio Control Protocol Using Uppaal". In *Journal of Logic and Algebraic Programming*, vol. 52-53, Holger Hermanns and Joost-Pieter Katoen (eds.), pp. 163–181, July-August, 2002.

[BLN03] Beyer, D., Lewerentz, C., and Noack, A. Rabbit: A tool for BDD-based verification of real-time systems. In *Computer Aided Verification*, pp. 122–125, Springer, 2003.

[Bozga] Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S. "Kronos: A model-checking tool for real-time systems". In *Computer Aided Verification*, pp. 546–550, Springer, 1998.

[Bos99] D. Bošnački. "Digitization of Timed Automata". In *Proceedings of the Fourth International Workshop on Formal Methods for Industrial Critical Systems (FMICS'99)*, Trento, Italy, pp. 283–302, 1999.

[BoD98a] D. Bošnački and D. Dams. "Integrating Real Time into Spin: A Prototype Implementation". In *Proceedings of the Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE/PSTV'98)*, Kluwer, pp. 423–439, 1998.

[BoD98b] D. Bošnački and D. Dams. "Discrete-Time PROMELA and Spin". In *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, LNCS 1486, Springer, pp. 307–310, 1998.

[ChH04] Chun K.Y., Hung D.V. "Verifying Real-Time Systems using Untimed Model Checking Tools". In *Technical report UNU-IISTTR-3002*, The United Nations University, International Institute for Software Engineering, 2004.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. "Model Checking". The MIT press, 1999.

[CHR91] Z. Chaochen, C.A.R. Hoare, and A. P. Ravn. "A Calculus of Duration." In *Information Processing Letters*, vol. 40, pp. 269-276, 1991.

[DaS95] J. Davies, and S. Schneider. "A Brief History of Timed CSP". In *Theoretical Computer Science*, vol. 138, number 2, pp. 243–271, 1995.

[DuS04a] B. Dutertre and M. Sorea. "Timed Systems in SAL". Technical Report, Computer Science Laboratory, SRI International, 2004.

[DuS04b] B. Dutertre and M. Sorea. "Modeling and Verification of a Fault-Tolerant Real-Time Startup Protocol using Calendar Automata". In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, LNCS 3253, Springer-Verlag, pp. 199–214, 2004.

[HMP92a] T. A. Henzinger, Z. Manna, and A. Pnueli. "What Good Are Digital Clocks?". In *Proceedings of the 19th International Colloquium on Automata, Languages, and Programming (ICALP'92)*, LNCS 623, Springer-Verlag, pp. 545–558, 1992.

[HMP92b] T. A. Henzinger, Z. Manna, and A. Pnueli. "Timed transition Systems". In *Proceedings of the Real-Time: theory in Practice, REX Workshop* . J. W. Bakker, C. Huizing, W. P. Roever, and G. Rozenberg, (Eds.) LNCS 600, Springer-Verlag, pp. 226–251, 1992.

[Hol93] G. J. Holzmann. "The Spin Model Checker, Primer and Reference Manual". Addison-Wesley, 2003.

[LeS97] D. Lesens and H. Sadi. "Automatic Verification of Parameterized Networks of Processes by Abstraction." In *International Workshop on Verification of Infinite State Systems (INFINITY'97)*, 1997.

[MRS03] L. M. Moura, H. Ruess, and M. Sorea. "Bounded Model Checking and Induction: From Refutation to Verification". In Proceedings of *Computer-Aided Verification*, LNCS 2725, Springer-Verlag, pp. 14–26, 2003.

[MOR04] L. M. Moura, S. Owre, H. Ruess, J. M. Rushby, N. Shankar, M. Sorea, and A. Tiwari. "Sal 2". In *Proceedings of International Conference on Computer-Aided Verification (CAV'04)*, LNCS 3114, Springer, pp. 496–500, 2004.

[NiS94] X. Nicollin, and J. Sifakis. "The Algebra of Timed Processes, ATP: Theory and Application." In *Information and Computation*, vol. 114, number 1, pp. 131–178, 1994.

[Pik05] L. Pike. "Real-Time System Verification by k-Induction". Technical report, NASA Langley Research Center. TM-2005-213751, 2005. Available at http://www.cs.indiana.edu/∼lepike/pub_pages/reint.html

[Pnu77] A. Pnueli. "The Temporal Logic of Programs". In *Proceedings of the 18th Annual Symposium of Foundations of Computer Science*, IEEE Computer Society Press, pp. 46–57, 1977.

[OsN96] J. Ostro and H. Ng. "Verifying Real-Time Systems with Standard Tools." In *AMAST workshop on real-time systems*, 1996. Available at http://www.cse.yorku.ca/∼stateclock/StateClock/amast.pdf

[PXZ02] A. Pnueli, J. Xu, and L. Zuck. "Liveness with (0,1,infinity)-Counter Abstraction." In *Proceedings of International Conference on Computer-Aided Verification (CAV'02)*, LNCS 2404, Springer-Verlag, pp. 107–122, 2002.

[Rus00] J. Rushby. "Verification Diagrams Revisited: Disjunctive Invariants for Easy Verification". In *Proceedings of International Conference on Computer-Aided Verification (CAV'00)*, LNCS 1855, Springer-Verlag, pp. 508–520, 2000.

[SaR06a] I. Saha and S. Roy. "A Finite State Modeling of AFDX Frame Management using Spin". In *Proceedings of the 11th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'06)*, 2006.

[SaR06b] I. Saha and S. Roy. "A Finite State Analysis of Time-triggered CAN (TTCAN) Protocol using Spin". In *Proceedings of the International Conference on Computing: Theory and Application (ICCTA'07)*, IEEE Computer Society, 2007.

[SMR07] I. Saha, J. Misra, S. Roy. "Timeout and Calendar based Finite State Modeling and Verification of Real-Time Systems". In *Proceedings of Automated Technology for Verification and Analysis (ATVA'07)*, LNCS 4762, pp. 284–299, 2007.

[Ste05] Steiner, W. "Model-Checking Studies of the FlexRay Startup Algorithm". Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, Research Report, 57/2005, 2005.

[StP02] W. Steiner and M. Paulitsch. "The Transition from Asynchronous to Synchronous System Operation: An Approach for Distributed Fault-Tolerant System". In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, IEEE Computer Society, pp. 329–336, 2002.

[SRS04] Steiner, W. and Rushby, J. and Sorea, M. and Pfeifer, H. "Model Checking a Fault-Tolerant Startup Algorithm: From Design Exploration To Exhaustive Fault Simulation". In *Proceedings of DSN*, 2004.

[TrC96] S. Tripakis and C. Courcoubetis. "Extending PROMELA and Spin for Real Time". In *Proceedings of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'96)*, LNCS 1055, Springer Verlag, pp. 329–348, 1996.

[Jia98] Jiacun Wang. "Timed Petri Nets: Theory and Application." Kluwer Acadamic Publishers, USA, October 1998.

[VaW86] M. Y. Vardi and P. Wolper. "An Automata-Theoretic Approach to Automatic Program Verification". In *Proceedings of 1st IEEE Symposium on Logic in Computer Science (LICS'86)*, IEEE Computer Society Press, pp. 332-344, 1986.